NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ACTIVE QUEUE MANAGEMENT MECHANISMS FOR REAL-TIME TRAFFIC IN MANETS

by

Leonidas Fountanas

December 2001

Chairman of Committee and Supervisor	Murali Tummala
Committee Members:	Robert Ives
	Robert Parker

Approved for public release; distribution is unlimited.

Report Documentation Page

Report Date 19 Dec 2001	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Active Queue Management Mechanisms for Real-Time Traffic in Manets	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Fountanas, Leonidas	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Naval Postgraduate School Monterey, California	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 126	

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2001	**3. REPORT TYPE AND DATES COVERED** Engineer's Thesis

4. TITLE AND SUBTITLE: Active Queue Management Mechanisms for Real-Time Traffic in MANETs.	**5. FUNDING NUMBERS**
6. AUTHOR(S) Leonidas Fountanas	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) SPAWARSYSCEN, D841 (Attn: Dr. North) 53560 Hull Street, San Diego, CA 92152 - 5001	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

This thesis develops active queue management mechanisms for real-time traffic for MANETs. Providing QoS for real-time applications is still an open issue as stated in RFC 2309. The proposed packet-dropping algorithm called Selective Early Discard (SED) selectively drops packets in order to spread the packet losses in a queue. Two variations of SED are also examined: one adds priority in order to provide service differentiation and the other utilizes timestamps to enable the intermediate nodes to drop packets that are likely to be unusable by the receiver due to excessive delay. Another scheme that drops bits instead of packets is also investigated.

Using simulation, the new queuing schemes are evaluated in a MANET environment, and their performance is compared with other existing QoS schemes, such as Random Early Discard (RED) and First In First Out (FIFO). Results indicate that SED minimizes the burst errors due to buffer overflow, thereby improving the performance for real-time traffic. SED is also capable of providing service differentiation; additional performance improvement can be realized by utilizing timestamps. Bit-dropping techniques can provide further performance improvements by spreading the error at the bit level (versus spreading the error at the packet level as in SED).

14. SUBJECT TERMS Joint Tactical Radio System, Network Simulator 2, Dynamic Source Routing, Quality of Service, Differentiated Services, Mobile Ad-hoc Network, real-time traffic, packet dropping, bit dropping, Voice over IP	**15. NUMBER OF PAGES** 126
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ACTIVE QUEUE MANAGEMENT MECHANISMS FOR REAL-TIME TRAFFIC IN MANETS

Leonidas Fountanas
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, June 1993
B.S. Business Administration, University of the Aegean (Greece), May 1999

Submitted in partial fulfillment of the
requirements for the degrees of

ELECTRICAL ENGINEER
and
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2001

Author: [illegible]
Leonidas Fountanas

Approved by: [illegible]
Murali Tummala, Chairman of Committee and Supervisor

[illegible]
Robert Ives, Committee Member

[illegible]
Robert E. Parker, Committee Member

[illegible]
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis develops active queue management mechanisms for real-time traffic for MANETs. Providing QoS for real-time applications is still an open issue as stated in RFC 2309. The proposed packet-dropping algorithm called Selective Early Discard (SED) selectively drops packets in order to spread the packet losses in a queue. Two variations of SED are also examined: one adds priority in order to provide service differentiation and the other utilizes timestamps to enable the intermediate nodes to drop packets that are likely to be unusable by the receiver due to excessive delay. Another scheme that drops bits instead of packets is also investigated.

Using simulation, the new queuing schemes are evaluated in a MANET environment, and their performance is compared with other existing QoS schemes, such as Random Early Discard (RED) and First In First Out (FIFO). Results indicate that SED minimizes the burst errors due to buffer overflow, thereby improving the performance for real-time traffic. SED is also capable of providing service differentiation; additional performance improvement can be realized by utilizing timestamps. Bit-dropping techniques can provide further performance improvements by spreading the error at the bit level (versus spreading the error at the packet level as in SED).

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis advisor, Dr. Murali Tummala, for all the fruitful and stimulating discussions and the guidance he provided in order to complete this thesis. I have consistently benefited from his professionalism and high standards.

I would like also to thank my committee members, Dr. Robert Ives and Dr. Robert E. Parker, for their advices and revision of this document.

Special thanks to past members of our research group Lt Leonardo Mattos (Brazilian Navy), Maj Kevin Shea (USMC) and Captain (USMC) Tyrone Theriot for their valuable discussions.

Finally, I would like to thank my parents and my friend Fani for their encouragement and support.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ABBREVIATIONS

AAL2	ATM Adaptation Layer 2
ABR	Associative Based Routing
ACK	Acknowledgment
ADPCM	Adaptive Differential Pulse Code Modulation
AODV	Ad Hoc On-Demand Distance Vector Routing
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
bps	Bits per second
BER	Bit Error Rate
C4I	Command, Control, Communications, “Computers” and Intelligence
CBR	Constant Bit Rate
CELP	Code-Exited Linear Prediction
COTS	Commercial-off-the-shelf
CQ	Custom Queuing
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CTS	Clear-to-Send
CU	Currently Unused
DARPA	Defense Advanced Research Projects Agency
DCF	Distributed Coordination Function
DiffServ	Differentiated Services
DMR	Digital Modular Radio

DoD	Department of Defense
DSCP	Differentiated Service Code Point
DSDV	Destination Sequenced Distance Vector Routing
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
FDMA	Frequency Division Multiple Access
FIFO	First-In First-Out
FTP	File Transfer Protocol
FS	Federal Standard
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol
IR	Infrared
ISI	Information Sciences Institute
JTRS	Joint Tactical Radio System
LAN	Local Area Network
LBNL	Lawrence Berkeley National Laboratory
LOS	Line of Sight
LSA	Link State Advertisement
LSP	Line Spectrum Pair
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network

MPLS	MultiProtocol Label Switching
NPS	Naval Postgraduate School
NAM	Network Animator
NS2	Network Simulator 2
OSI	Open Systems Interconnection
OTcl	Object Tool command language
PARC	Palo Alto Research Center
PHB	Per Hope Behavior
PQ	Priority Queuing
QDR	Quadrennial Defense Review
QoS	Quality Of Service
RED	Random Early Discard
RF	Radio Frequency
RFC	Request For Comments
RREP	Route Reply
RREQ	Route Request
RSVP	Resource Reservation Protocol
RTP	Real-time Transport Protocol
SAR	Search and Rescue
SD	Selective Dropping
SDR	Software Defined Radio
SED	Selective Early Dropping
TCL	Tool Command Language
TCP	Transmission Control Protocol

TDMA	Time Division Multiple Access
TORA	Temporally Ordered Routing Algorithm
TOS	Type of Service
UCB	University of California at Berkeley
UCLA	University of California at Los Angeles
UDP	User Datagram Protocol
UHF	Ultra High Frequency
USC	University of Southern California
VBR	Variable Bit Rate
VINT	Virtual InterNetwork Testbed
VoIP	Voice over Internet Protocol
VTC	Video Teleconferencing
WAN	Wide Area Network
WFQ	Weight Fair Queuing
WLAN	Wireless Local Area Network
WRP	Wireless Routing Protocol
ZRP	Zone Routing Protocol

EXECUTIVE SUMMARY

Today’s demands on military operations call for extensive use of digital communication devices to support real-time traffic in the battlefield. The Joint Tactical Radio System (JTRS) acquisition program was intended for all the services to combine and integrate all tactical radio developments into one platform. These communication devices will operate in a Mobile Ad hoc Network (MANET) environment and are capable of transmitting voice, video and data.

MANETs present many challenges, especially when real-time traffic must be supported in terms of providing Quality of Service (QoS) guarantees. Providing QoS for real-time traffic over IP-based networks is still an open issue because existing active queue management schemes have been designed for TCP-compatible traffic. MANETs present the worst-case scenario for QoS guarantees due to their distinct characteristics, such as contention from multiple users (when using 802.11) and limited bandwidth. The objective of this thesis is to develop new active queue management schemes for MANETs that are more efficient compared with existing algorithms. These schemes are based on packet and bit dropping techniques.

A packet-dropping algorithm, called Selective Early Discard (SED) that selectively drops packets in order to reduce the burst error, is developed in this thesis. Two variations of SED are also examined: one adds traffic priority in order to provide service differentiation and the other utilizes timestamps to enable the intermediate nodes to drop packets that are likely to be unusable by the receiver due to excessive delay. Another scheme that drops bits instead of packets is also investigated.

Using simulations, the new queuing schemes are evaluated in a MANET environment, and their performance is compared with other existing QoS schemes, such as Random Early Discard (RED) and First In First Out (FIFO). The performance metrics used for evaluation of the QoS schemes are packet loss, average end-to-end delay and distribution of packet losses.

The simulation results indicate that SED minimizes the burst errors due to buffer overflow, thereby improving the performance for real-time traffic. SED is also capable of providing service differentiation by utilizing the services of DiffServ in which packets are marked as high and low priority. Using timestamps, additional performance improvements of the proposed QoS schemes are realized as unusable packets at the destination due to excessive delay are dropped in intermediate nodes. Bit-dropping techniques can provide further performance improvements by spreading the error at the bit level (rather than spreading the error at the packet level as in SED).

I. INTRODUCTION

A. MOTIVATION

Today's demands on military operations call for extensive use of digital communication devices to support real-time traffic in the battlefield. As a result, the military is very interested in the development of such devices as the Software Defined Radios (SDR) for the next generation tactical communications. The military term for these radios is Joint Tactical Radio Systems (JTRS).

The JTRS program was intended for all the services to combine and integrate all tactical radio developments into one platform. The goal of the JTRS project is to develop a family of affordable, interoperable high-capacity, tactical software defined radios, providing both line-of-sight and beyond-line-of-sight Command, Control, Communications, Computers and Intelligence (C4I) capabilities to the war fighters [1]. This family of radios will cover an operating spectrum from 2 to 2,000 MHz and is capable of transmitting voice, video and data.

Many wireless networking problems have to be solved for the efficient design and deployment of these communications devices that operate in a Mobile Ad-hoc Network (MANET) environment. Among these problems, the provision of Quality of Service (QoS) is one of the most important for two reasons: (1) providing QoS for real-time traffic over IP-based networks is still an open issue, and (2) MANETs due to their distinct characteristics present the worst case scenario for QoS guarantees.

Providing QoS for streaming applications is still an open issue as stated in RFC 2309 [28]. The existing QoS schemes have been designed for Transport Control Protocol (TCP)-compatible applications and, as a result, they do not take into account the specific characteristics of the real-time traffic, such as voice and video. The Random Early Discard (RED) algorithm [29] is the recommended Internet Engineering Task Force (IETF) queue management scheme for congestion avoidance in TCP-compatible connections. It is widely used as evidenced by its implementation in the latest versions of Cisco routers [24]. Although RED is the best existing solution, it does not achieve the goal of providing the required *QoS guarantees* in future networks in which the majority of the traffic consists of flows that are unresponsive to congestion notification or responsive but more aggressive than TCP [28].

MANETs present many challenges: they are bandwidth limited; there is contention from multiple users; and the nodes send, receive and relay packets. Moreover, they are focused on real-time traffic delivery, thus there is a need for providing better treatment for some sources.

Based on the above considerations the motivation behind this thesis is to investigate new active queue management QoS schemes for MANETs that take into account the specific characteristics of real-time traffic.

B. OBJECTIVES

The objectives of this thesis are to conduct an in-depth study in providing QoS guarantees for real-time traffic in MANETs and to propose new active queue management schemes that are more efficient compared with existing algorithms. This work is limited to real-time traffic, such as voice and video, and assumes that the non-real-time traffic, such as e-mail and ftp, is handled separately. More specifically, the research goals can be summarized as follows:

- Develop new active queue management algorithms that selectively drop packets or bits to spread the packet losses in a queue, thereby improving the performance of real-time traffic in MANETs.
- Extend the proposed algorithms by adding priority and timestamps in order to achieve service differentiation between high and low priority traffic sessions.
- Conduct simulation runs to compare the performance of the proposed algorithms with that of Random Early Discard (RED) and First In First Out (FIFO) queuing schemes.

C. ORGANIZATION OF REPORT

The thesis is organized as follows. Chapter II discusses the main issues in MANETs by emphasizing the aspects that play an important role in providing QoS in these networks. Chapter III gives a detailed analysis of the real-time traffic characteristics and a review of the existing QoS protocols and schemes. Chapter IV describes the packet-dropping algorithms: the Selective Early Discard (SED) and the extension of SED using IN/OUT packets (SED/IO). In addition, a variation of these algorithms using timestamps is presented. Chapter V presents a bit-dropping scheme and a demonstration of its effectiveness using the Federal Standard 1016 Code- Excited Linear Prediction (CELP) codec. Chapter VI describes the Network Simulation 2 (NS2) package and presents simulation results for the proposed algorithms along with RED and FIFO queuing schemes. Chapter VII summarizes the results and provides conclusions and recommended future work. Appendix A contains a segment of the SED and SED/IO code files used in the simulations.

THIS PAGE INTENTIONALLY LEFT BLANK

II. MOBILE AD HOC NETWORKS (MANET)

A. OVERVIEW

Wireless networking appeared in the 1970s when these networks were called packet radio networks [2]. Since then, mobile wireless networks have developed into two main technologies: mobile IP networks and Mobile Ad-Hoc Networks (MANETs).

Figure 1 shows the conceptual differences between the existing wireless networks today. First, the mobile IP networks or cellular networks consist of fixed, wired gateways known as base stations. A mobile host within these networks communicates with the nearest base station. One of the major problems related to cellular networks is called “handoff, ” which is the process of transferring a mobile station from one channel or base station to another without noticeable delay or packet loss. Another problem is the mobile nodes are able to connect to the network only if the base station is within its communication range, thus nodes are limited to places where such a cellular infrastructure exists. In contrast, MANETs do not rely on pre-existing infrastructure, such as base stations. They are self-organizing wireless networks consisting of a number of mobile nodes. Each mobile node in a MANET can send, receive or forward a packet [3], [4].

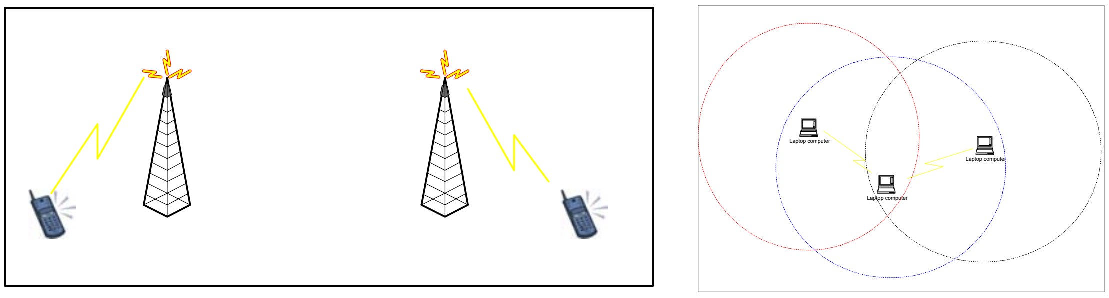


Figure 1. Existing Mobile Communication Networks: (a) Mobile IP (cellular) and (b) Mobile Ad-hoc Networks.

MANETs have four distinct characteristics [3]: dynamic topologies, bandwidth constraints, energy-constraints and limited physical security. The first characteristic allows the nodes to move arbitrarily and unpredictably causing possible failures in links or routes. The second concerns the wireless links typically having a significantly lower capacity than their wired counterparts. Moreover due to contention from multiple users, fading, noise and interference, the capacity is highly time variable. Third, the nodes are usually battery-operated; therefore, management of the power is needed. Finally, wireless links are in general vulnerable to security threats like eavesdropping, spoofing, and denial-of-service attacks. Together these characteristics pose a challenge in providing quality of service.

Currently, one of the areas of interest in mobile ad hoc networks is the provision of *QoS guarantees*. The first aspect of QoS is related to routing for which much research has been done and many different routing protocols have been proposed in the current literature. Secondly, QoS is affected by the Medium Access Control protocol (MAC). Although the most commonly used MAC protocol in MANETs is the 802.11, it appears to be unsuitable, especially under high traffic loads. In this thesis QoS algorithms are developed for real-time traffic over IP-based networks. The algorithms are then applied to a MANET environment as a means to evaluate the proposed algorithms as these networks present the worst-case scenario for providing *QoS guarantees*.

The MANET routing protocols must guarantee compatibility and interoperability with Internet standards in the other layers [3]. A MANET node may act as a source if the traffic is being originated within the node or as a relay if it is an intermediate node. The proposed protocol stack from the IETF MANET working group for a mobile node is depicted in Figure 2. Each packet is sent to the wired or wireless MAC protocol in order to be forwarded via the wired or wireless network interface, respectively, to the next hop.

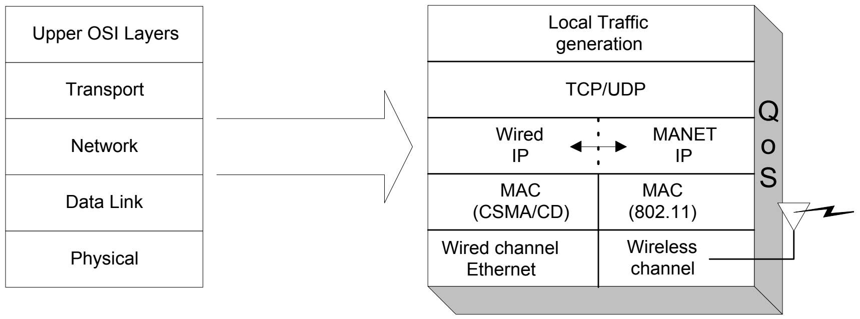


Figure 2. Mobile Node Protocol Stack (From ref. [14]).

B. COMMUNICATION LINK DESIGN ISSUES

A wireless communication system must be able to provide reliable transmission of data using the lowest possible bandwidth and power. Wireless channels in which MANETs operate make the design of a wireless communication system a difficult task due to their distinct characteristics. Specifically, wireless channels are known for high error rates and limited bandwidth. In general, systems are designed for the worst-case propagation conditions; however, because of the unpredictability of radio channels, a system can also be designed to adapt to the link quality at both the link layer and the network layer level.

1. Wireless Channel Characteristics

The mobile radio channel is a difficult environment and can vary from simple Line-Of-Sight (LOS) to one that has obstructions like buildings, trees or mountains. The Friis formula for free space propagation gives the received signal strength when the transmitter and receiver have a clear line of sight path between them:

$$P(d) = \frac{P_t G_t G_\lambda \lambda^2}{(4\pi)^2 d^2 L} \tag{3.1}$$

where P_t is the transmitter power, d is the distance between the nodes, G_t and G_r are the transmitter and receiver antenna gains, respectively, L is the system loss factor not related to propagation ($L \geq 1$), and λ is the wavelength.

The actual transmission loss in LOS radio waves is different than the free space loss due to reflection, refraction and/or diffraction. A two-ray reflection model, which considers both the direct path and a ground reflection path, provides a better approximation model than the free-space model at large distances [3]. Several other propagation models are available in the literature that predict the large scale effects by taking into account such factors as diffraction and refraction. Also, there are outdoor propagation models that consider the terrain profile and estimate the path loss over irregular terrains. All these models are appropiate for the prediction of signal strength at a particular receiving point or in a specific area while varying widely in their approach, complexity and accuracy [3]. In this thesis, a two-ray propagation model using the crossover distance parameter is used.

2. Medium Access Control (MAC) [8]

In wireless networks, users share a common medium, thereby creating a need for a protocol that provides efficient and fair access. The most commonly used MAC protocols in wireless networks are Frequency Division Multiple Access (FDMA), Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). The CSMA/CA scheme, adopted in the IEEE 802.11 standard, is the most widely MAC protocol in wireless LANs and MANETs. In this thesis, 802.11 is used in network simulation.

In CSMA/CA, when a node wishes to transmit a packet, it first listens to the medium. If the channel is idle, it transmits the packet; otherwise, it waits for a random amount of time based on a "backoff factor." When the medium is idle, the transmitting node gradually decreases its backoff counter; however, if the medium is busy the counter is frozen. The packet is transmitted when the counter reaches zero. The Ready-to-Send (RTS) and Clear-to-Send (CTS) messages are used to reduce the collision problem, which occurs if two nodes try to access the medium simultaneously. When the backoff counter reaches zero, the transmitting node sends a RTS packet containing information about the length of the message that is ready for transmission. Then, if the receiving node hears the RTS, it will send a CTS packet, allowing the transmitter to send its packet. Finally, upon successful reception of the packet, the receiving node sends an ACK.

3. Performance Degradation in Ad Hoc Networks [5], [6], [7]

In Ad hoc networks, due to the interactions with the MAC layer, the performance in terms of throughput and end-to-end delay often degrades significantly, which can be attributed to hidden node, exposed node and control packet overhead. These problems cause throughput instability, unfairness, and dependence on the number of nodes, size of the area, and the length of the packets. These in turn affect the quality of service at the application layer level.

The hidden node and exposed node problems are not completely isolated in the IEEE 802.11 standard. In spite of using RTS, CTS, and random back off mechanism, collisions still happen. The result is degradation in throughput is referred to as throughput instability and unfairness [5].

Figure 3 illustrates the hidden node problem. Suppose that station A is transmitting to station B, and C is ready to transmit to B or another station. Station C is out of the range of A, hence C does not detect the carrier from A. As a result, station C transmits its message and a collision occurs. Station A is hidden from C.

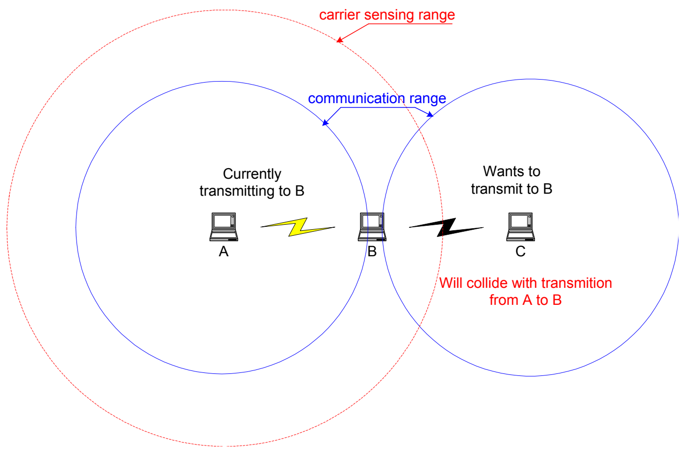

carrier sensing range
communication range
Currently transmitting to B
Wants to transmit to B
A
B
C
Will collide with transmition from A to B

Figure 3. Hidden node problem.

The exposed node problem exists in the CSMA/CD protocols because the carrier sensing range is larger than the communication range between two nodes. In the IEEE 802.11 standard, the required signal-to-noise ratio (threshold) for carrier sensing is lower than the corresponding range for error free reception, which is the communication range. Figure 4 depicts the exposed node problem. Station B transmits to A. Station C is within the transmission range of B and wants to transmit to D. Although the medium is free near Station C, it cannot send to D because C detects B's carrier.

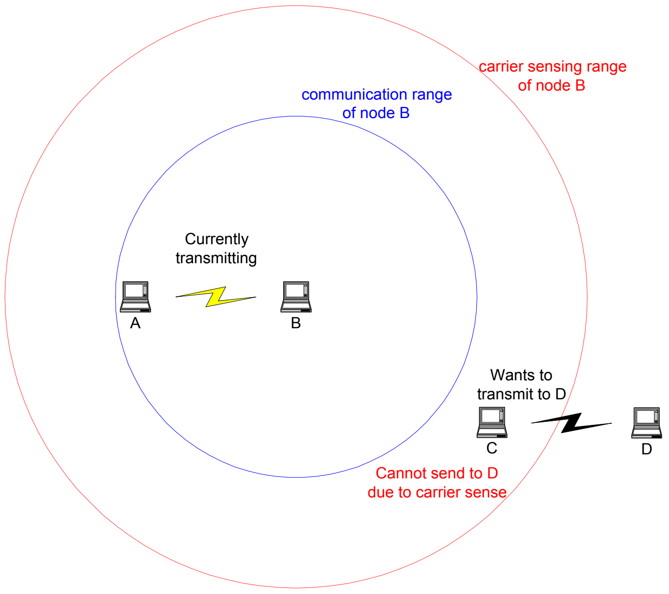


Figure 4. Exposed node problem.

Throughput degradation is also experienced in ad hoc networks by either increasing the number of nodes within a specific area or decreasing the packet size [6]. The degradation in throughput as the packet size becomes smaller is due to increased overhead. Each RTS packet is 40 bytes long while CTS and ACK packets are 39 bytes and the MAC header is 47 bytes long. As the number of nodes within a given area is increased, the throughput decreases because the hidden and exposed node problems become more pronounced.

This section summarized the problems related to the 802.11 MAC protocol when used in ad hoc networks. Both TCP and UDP connections are affected by these problems.

Additionally, considering that UDP is unresponsive to congestion notification, its performance is expected to be worse than that of TCP.

C. NETWORK LAYER ISSUES

Routing, a function associated with Layer 3 (OSI model), is a technique used by the network to determine a path for packets from a source to a destination. In each node, a router examines the packet's destination address, estimates the best path and forwards the packet along this route. The routing information is related to the topology and conditions of the network. In a MANET in which the topology changes frequently, the routing information needs to be updated more frequently than in the fixed networks.

1. Conventional Routing Protocols [12]

There are two types of widely used routing protocols in packet switched networks: link state and distance vector routing algorithms. In link-state routing, each router maintains a database that describes the topology of the entire network with a cost for each link. Whenever the network topology is changed, a message known as Link State Advertisement (LSA) floods throughout the network. The nodes take the information and update their database by using the shortest path algorithm, usually Dijkstra's, to estimate the next hop for each destination.

In distance vector routing, each node informs its neighbors of its routing table by periodically broadcasting an estimate of the shortest distance to every other node in the network. Each router, as it receives an update for each destination in each table, compares the metric in its table with that in the neighbor's table plus the cost of reaching that neighbor. These protocols are based on the distributed Belman-Ford routing algorithm.

Link state protocols compared to distance vector are more stable, have faster convergence and discover more easily a network topology. On the other hand, distance vector protocols are easier to implement and require less memory. Finally, the associated overhead in distance vector protocols is constant regardless of the amount of topology changes in the network.

2. Overview of Ad hoc Routing Protocols [12], [13]

Due to the distinct characteristics of MANETs, the design of an efficient routing protocol is a challenging task. The main reason is the traditional routing protocols described above are designed for a relatively stable network topology. In addition, the conventional protocols rely on some form of distributed routing databases. In MANETs, routers cannot be assumed to have persistent data storage, and they cannot always be trusted [2].

The most common categorization of ad hoc routing protocols is presented in Figure 5. The basic difference between the on-demand and table-driven categories is related to the way the routing information is collected. On-demand routing protocols collect routing information only when needed by using the route discovery procedure. On the other hand, table-driven protocols constantly propagate routing information.

Table-driven routing protocols, such as Dynamic Destination Distance Vector (DSDV) [9, 11] and Wireless Routing Protocol (WRP), continuously evaluate routes (proactive). In contrast, on-demand or source-initiated protocols create routes only when needed (reactive). Protocols that belong to this category are Dynamic Source Routing (DSR), Temporary Ordered Routing Algorithm (TORA) and Associative Based Routing (ABR). Finally, hybrid routing protocols, such as Ad hoc On-demand Distance Vector (AODV) [11, 14] and Zone Routing Protocol (ZRP) [10, 15], have both proactive and reactive characteristics.

The DSDV is the only suitable proactive protocol when a reasonable time is allowed in order to converge. DSR and AODV provide the best performance in most scenarios according to simulation studies [12]. Notably, DSR outperforms AODV in lower traffic density and mobility and when the number and size of the network is low (e.g., less than 20 nodes). The main drawback of DSR that leads to performance degradation in large or multihop networks is the need to include the entire route in each packet. ZRP divides the network into zones/clusters and provides a good solution for large networks using a reactive approach for routing between the zones and proactive approach within a zone.

Clearly none of the proposed ad hoc routing protocols provide the best performance in all scenarios. Certain protocols are well suited for specific situations. In this thesis, the DSR protocol is used in the simulation model because the network size is small, less than 20 nodes, and the mobility is usually low [11]. DSR is described in the following subsection.

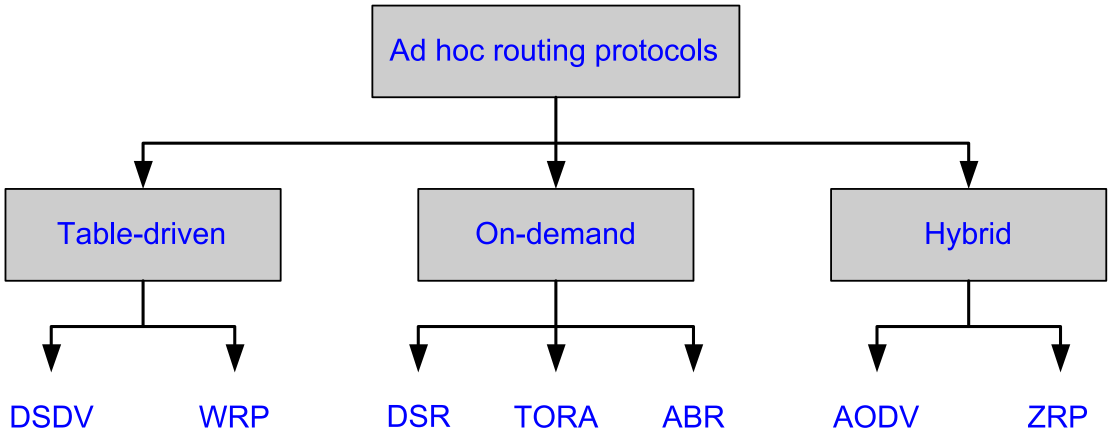
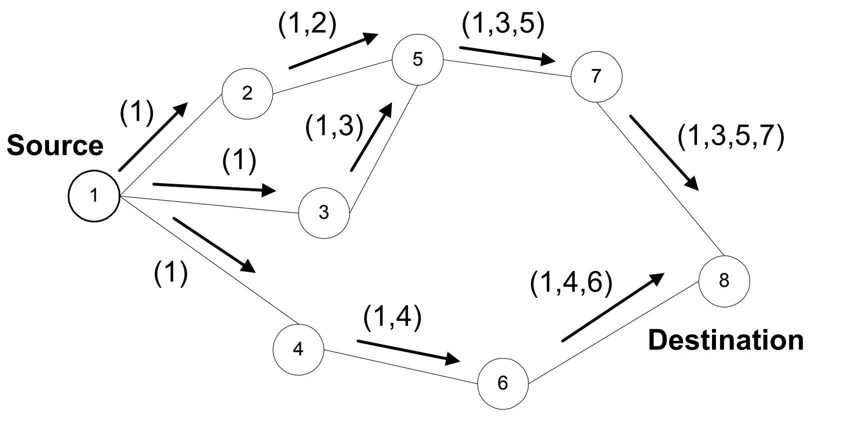


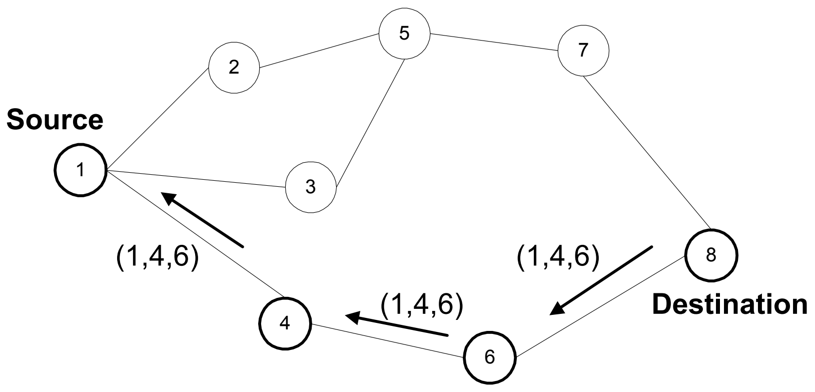
Figure 5. Classification of ad hoc routing protocols.

3. Dynamic Source Routing (DSR) [12], [13]

DSR is a reactive protocol, and its operation is based on source routing. In source routing, each packet carries a list of the nodes leading to the destination in its header. Each node updates its route cache whenever a new route is learned. If a source node has a packet to send and no route is available to the destination in its cache, it floods the network with a route request (RREQ) packet, which contains the address of both the sender and the destination. The nodes that receive the RREQ, if they do not know a route to the destination, forward the RREQ packet after adding their own address to the list. In order to reduce the control packets in the network, a DSR node does not forward a RREQ packet when it finds its own address in the list. Finally, a reply packet is sent back to the source by either an intermediate node, which knows a route to the destination or the destination. Figures 6 (a) and (b) illustrate this route discovery process.



(a) Building route record during route discovery



(b) Propagation of route reply with source route record

Figure 6. Creation of Route Cache in DSR (After Ref. [15]).

Once the route is discovered, route maintenance is accomplished through the use of route error packets and acknowledgements. An error packet message is sent back to the source when a route is broken and the nodes update its route cache; otherwise, an acknowledgement verifies that the next node received the packet.

D. SUMMARY

This chapter introduced mobile ad hoc networking concepts in order to provide the necessary background. It discussed issues in MANETs, such as routing and medium access control. The next chapter provides a detailed discussion of the QoS for real-time traffic and a review of the existing active queue-management QoS schemes.

THIS PAGE INTENTIONALLY LEFT BLANK

III. QUALITY OF SERVICE (QOS) ISSUES

Mobile ad hoc networks, as mentioned in the previous chapter, are generally the worst-case scenarios for providing *QoS guarantees*. Not only is the performance of these networks unpredictable due to network dynamics, but they also operate with a limited-bandwidth, in a high- error-rate environment. Additionally, if a MANET supports real-time traffic, there is a need for effective traffic management by implementing an efficient QoS scheme. This chapter discusses the characteristics of real-time traffic with emphasis placed on voice traffic. Also, the chapter reviews the existing approaches for providing *QoS guarantees*. Based on this discussion, the next two chapters will develop some new QoS schemes.

A. REAL-TIME TRAFFIC CHARACTERISTICS

Real-time traffic consists of multimedia applications like audio and video conferencing, video-on-demand, distributed interactive applications and network games; applications, such as FTP and e-mail, are considered as non-real time-traffic.

1. Requirements for Real-Time Applications

Real-time traffic applications can be divided into three categories according to their traffic profile [17]: continuous data sources in which fixed-size packets are generated at constant intervals; on-off sources in which fixed size packets are generated at fixed intervals with the source alternating between active and inactive periods; and finally, variable packet size in which the source produces variable-length packets at uniform intervals. Real-time simulations, audio conferencing and digitized video with different compression ratios are three examples of the different traffic profiles, respectively.

The bandwidth requirements are different for different applications. For a continuous data source, the required bandwidth is usually large (1-10 Mbps) and can be made available when needed. In general, multimedia data are compressed and encoded in order to reduce the redundancy of data. This results in variable bit rate (VBR) data, hence the bandwidth demand of an application varies over time. In VBR, there is a large difference between the peak and minimum data rate. Due to bandwidth limitations of wireless networks, bandwidth reservation at peak-rates leads to poor utilization.

Real-time applications are delay sensitive in both end-to-end delay and delay variation (jitter). End-to-end delay is the total delay experienced by a packet and consists of the compression (decompression), packetization (depacketization), propagation and queuing delays. The only random component in the end-to-end delay is the queuing delay, which results in the non-uniform arrival of packets at the destination (jitter). The bounds on delay are dictated by the application. Packets that exceed the delay bounds are not usable at the receiver.

Usually, real-time applications can tolerate packet losses. The packet loss tolerance is dependant on the application and compression /packetizing schemes used. Additional discussion on voice traffic is presented in the following subsection as it is used in this thesis in the simulation studies.

2. Real-Time Voice Characteristics

Voice quality, the most important characteristic when voice is transmitted over the network, is characterized in terms of qualitative and quantitative measures. Qualitative measures are mainly voice fidelity and intelligibility while quantitative measures reflect the performance of the underlying transport mechanisms. The two most important performance metrics that affect voice quality are packet loss and end-to-end delay.

According to International Telecommunication Unit (ITU) recommendation G.114 [41], the one-way end-to-end delay for toll quality speech must be less than 150 ms. End-to-end delays of 150 to 300 ms cause degradation of voice quality but are still acceptable in international calls and satellite transmission. For delays of more than 300 ms, significant voice quality degradation occurs. Additionally, because voice is an isochronous application, the jitter should be small so that the play back at the receiver remains smooth. Jitter buffers are usually effective for a maximum jitter of 100 ms.

Voice traffic can tolerate a small amount of error due to packet loss, leading to slight degradation in quality. The amount of tolerable loss is different for various compression schemes and depends on application requirements. Generally speaking, the greater the bandwidth reduction due to compression, the more sensitive the coded voice packets are to packet losses. The ITU G.729 codec for toll quality constrains error due to packet loss to 1% in order to avoid audible errors. Experimental results reported by Nortel [33], Cisco [34] and other companies on voice over IP (VoIP) applications suggest that a much higher packet loss rate of 4% is acceptable without significant degradation in voice quality. The acceptable amount of packet losses is discussed in detail in the packet dropping study in Chapter IV.

3. Transport Protocol

Two transport protocols are available for the transmission of real-time traffic: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Real-time Transport Protocol (RTP) is an application layer protocol that supports real-time traffic when UDP is used.

TCP provides a reliable connection between the hosts and guarantees that packets are delivered in the same order in which they were sent. Reliable data transfer is achieved using acknowledgements and packet retransmissions. The source and the destination IP addresses and port numbers explicitly identify a TCP connection. TCP protects the data using a checksum and provides flow and congestion control. The TCP header is at least 20 bytes long. The optional field in the header is used for extensions to TCP as described in RFC 2018 [20] and RFC 768 [21].

User Datagram Protocol (UDP) provides connectionless unreliable service with low overhead. The UDP header length is 8 bytes and consists of four fields, each being two bytes in length. These fields are source and destination port numbers, length of the entire UDP segment, and an optional checksum.

TCP is rarely used in real-time traffic applications. It provides reliable transformation of data; however, this is not necessary for some real-time traffic applications because they are loss tolerant. TCP is a point-to-point protocol that sets up a connection between two end-points. Consequently, in multicast distribution for N participants, there is a need for N×N connections, thus increasing complexity as well as packet traffic. TCP retransmits the packets when losses occur, and a majority of the retransmitted packets may not be usable because of the additional delay. Window backoff occurs when TCP sources lower their rates in response to packet losses. However, real-time traffic is delay sensitive and, therefore, the sources cannot reduce the transmission rate because it often results in unacceptable packet delay at the destination.

UDP is a better choice than TCP for real-time traffic because it provides relatively lower complexity, multicast capabilities, lower overhead and no packet retransmissions. However, there are also limitations in using UDP. UDP does not provide packet delivery in order to identify duplicate packets and detect losses. RTP over UDP provides a solution to these.

RTP [22] is designed to handle end-to-end network transport functions for real-time applications by using sequence numbers and timestamps. The sequence numbers help solve the problems of packet loss, duplication and out-of-order delivery. The lost packets are detected and replaced by dummy packets while the out-of-order packets, if not too late, are reordered in a buffer. The timestamp is used for packet synchronization at the destination using a delay buffer. The Real-Time Transport Control Protocol (RTCP) is a companion protocol to RTP. Each connection participant periodically issues an RTCP packet to provide information about the quality of reception.

An important issue for the implementation of IP/UDP/RTP is the associated overhead, a total of 40 bytes (20+8+12). The problem becomes larger in the case of voice traffic due to small payload sizes, usually 20-30 bytes. However, the compression of IP, UDP and RTP headers on a link-by-link basis reduces the total overhead to two bytes when no UDP checksums are sent or four bytes when the checksums are sent [23]. The header compression helps RTP run more efficiently, especially over low speed links where both the associated overhead and transmission delay are reduced significantly. On the other hand, the compression of the packet header adds significant complexity in the case of multihop networks because each intermediate node has to decompress and then compress a packet before its transmission to the next node.

B. QOS OVERVIEW

Quality of Service (QoS) is defined as the collective effect of network performances that determines the degree of user satisfaction in the service [39]. The three fundamental pieces for QoS implementation are: QoS within a single network element (for example, queuing, scheduling, and traffic shaping), QoS techniques for coordinating QoS from end-to-end among network elements, and QoS policy, management, and accounting functions for controlling and administering end-to-end traffic across a network [24]. An autonomous network like a MANET is able to implement one or a combination of these QoS architectures.

When using techniques that provide a level of assurance for the network traffic, two QoS types exist: resource reservation and traffic prioritization. Resource reservation means that network resources are allocated according to an application's QoS request and are subjected to bandwidth policy. In traffic prioritization, the traffic is classified, and the network elements give preferential treatment to applications having a greater demand for the network resources [25].

1. QoS Protocols

Two widely used protocols for providing QoS are the Resource reSerVation Protocol (RSVP) and Differentiated Services (DiffServ).

RSVP [26] provides QoS by reserving resources, such as bandwidth, during the signaling process. An overview of how the protocol works is illustrated in Figure 7. The source sends a PATH message to the receiver(s) containing the traffic specification information, such as upper and lower bounds of bandwidth, delay and jitter. The receivers send a RESV message that consists of the traffic specification and a request specification containing the packets for which the reservation is being made along with the type of service. When each intermediate router receives the RESV message, it sends a request to the next router. If the request can be satisfied, the router sends the PATH upstream to the next router; otherwise, it returns an error back to the receiver. If the last intermediate router accepts the request, it sends a confirmation message back to the receiver and the path is established.

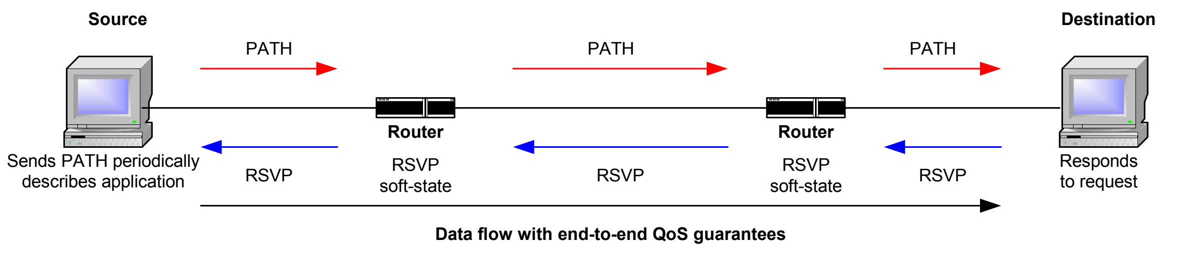


Figure 7. Resource reservation using the RSVP Protocol (After Ref. [26]).

RSVP allocates resources to individual flows, which can lead to scalability limitations since signaling information between the routers increases proportional to the number of flows. Two reasons make this solution unsuitable for MANETs. First, MANETs are bandwidth limited, and the associated overhead from control (signaling) packets may cause congestion in the network. Second, topology changes often cause established routes to fail [45].

In contrast, in DiffServ [27], the need for per-flow resource reservation as well as signaling in each router along a data path are eliminated. Traffic is divided into a small number of forwarding classes that have similar QoS requirements, and resources are allocated on a per-class basis. Most classification and policing are done at the network edge, and the classified and marked packets at the boundary of the network in ingress nodes receive a different Per-Hop forwarding Behavior (PHB) in interior nodes.

Figure 8 shows the block diagram of a packet classifier and traffic conditioner. The packet classifier selects packets in a traffic stream based on information in the packet header, such as the flow ID field in IPv6 or the type of service (TOS) field in IPv4.

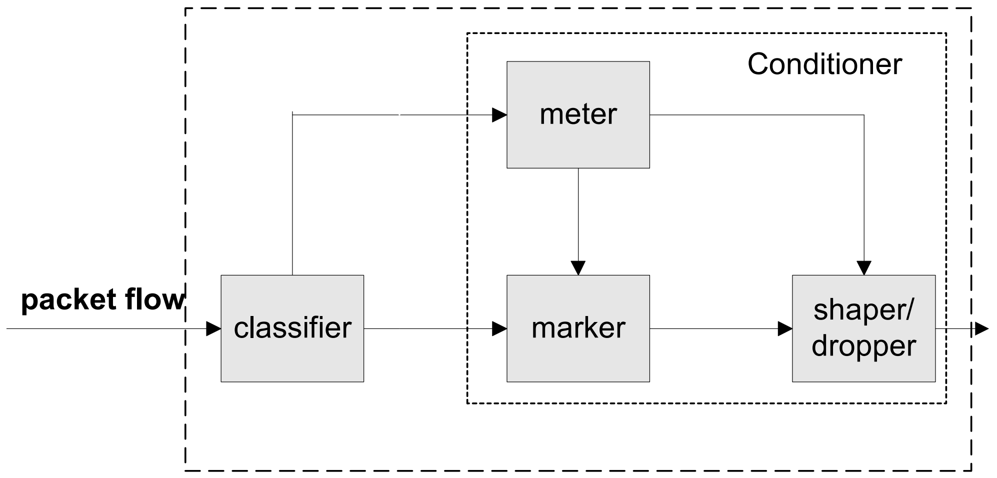


Figure 8. Block diagram of a packet classifier and traffic conditioner [After Ref. [27]).

After classification, the meter measures the traffic stream against a specific traffic profile while the marker manipulates the packet's Differentiated Services (DiffServ) field indicating that the packet has been added to a specific DiffServ behavior. The shaper delays packets of a traffic stream in order to conform to a specific traffic profile; therefore, when the shaper's buffer is filled, packets are discarded.

In MANETs, nodes serve the dual roles of router and source, which makes the distinction between *ingress* nodes and *interior* nodes complicated. Also, additional protocol processing is needed to carry out these dual roles, which in turn could lead to draining of the battery power. Consequently, a QoS protocol with low overhead is desirable for MANETs [14], [46].

RSVP and DiffServ protocols are designed to provide QoS under specific environments and applications. The highest level of QoS is provided by RSVP because it reserves the necessary bandwidth for a partial number of flows. However, this is achieved at the price of complexity and overhead. On the other hand, the overhead in DiffServ is low. DiffServ methods are characterized by their simplicity because the prioritization of packets is feasible using simple algorithms and flexibility because DiffServ is able to identify specific applications and determine different traffic profiles. Thus, an effective implementation of RSVP protocol is a very difficult task in MANETs. On the other hand, DiffServ policies are more attractive since no signaling is necessary and the overhead is low.

2. QoS Schemes

The implementation of QoS protocols involves specific rules based on queuing algorithms that sort the arriving packets and/or prioritize them onto output links. These algorithms, called congestion control algorithms, can be divided into two categories: queue management and scheduling. Queue management QoS algorithms drop packets when necessary or appropriate in order to manage the length of the queue. Scheduling algorithms manage the allocation of the bandwidth among flows by determining which packet to forward next [28].

a. Scheduling Algorithms

There are a number of scheduling algorithms proposed in the literature to make the Internet a QoS-capable network [28]. Each queuing algorithm has been designed to solve a specific network traffic problem and has a particular effect on network performance. The following schemes are widely used for scheduling.

First-in-First-out (FIFO), the simplest queuing scheme, lets the packets leave the queue in the order of their arrival. It also accepts packets until the queue is full and then drops the incoming packets.

Custom Queuing (CQ) guarantees bandwidth at a potential congestion point by reserving a specific portion of the available bandwidth for one or more traffic flows. The remaining bandwidth is used to serve the other traffic.

Weighted Fair Queuing (WFQ) categorizes traffic flows into high and low priority, based on volume of packets seen by a router or switch. Low-bandwidth flows are served first, and the remaining bandwidth is shared among the high bandwidth flows according to assigned weights. As a result, WFQ favors low-bandwidth traffic.

Priority Queuing (PQ) provides better treatment of some packets by serving them first. For example, in the case of two kinds of traffic, high and low priority, the high priority packets are served first.

b. Active Queue Management Schemes

Random Early Detection (RED) is the most widely used queue management scheme for congestion avoidance. The Internet Engineering Task Force (IETF) recommends implementing the RED queuing scheme as the best solution to improve performance in the Internet [28].

The RED algorithm was initially proposed by Floyd and Jacobson [29] as an effective mechanism to control congestion in a network. The main goal of the RED algorithm is to avoid congestion rather than react to it. RED achieves this by detecting the onset of congestion in order to maintain the network in a region of low delay and high throughput [17]. In general, RED drops packets randomly with increased probability as the queue size grows. Additionally, RED algorithm improves the performance in TCP-compatible flows by solving the global-synchronization problem and by reducing the burst errors due to buffer overflow. Specifically, it takes advantage of the back-off mechanism in TCP by dropping packets at random when traffic exceeds a predetermined threshold. This causes one TCP connection at a time to back off, reducing the congestion. Therefore, the back-off of almost all TCP connections, known as global-synchronization problem, is avoided. Additionally, RED reduces the burst errors due to buffer overflow because it drops packets randomly instead of the newly arriving packets. However, in the case of flows that are unresponsive to congestion notification, like real-time traffic using UDP, RED does not solve the problem with burst errors because it cannot control the source rate. Dropping packets early increases the total error compared to the error produced by the FIFO queuing scheme, which allows the smallest possible error for unresponsive flows [28].

The RED algorithm is described in [29] and works as depicted in Figure 9. For each arriving packet, a time-based average queue length is first computed. The algorithm has three congestion states: normal, congestion avoidance and congestion control; and two thresholds, Θ_{min} and Θ_{max}. If the average queue length is less than Θ_{min}, no packets are dropped. If the average queue length is between the Θ_{min} and Θ_{max}, a randomly chosen packet from the queue is dropped with probability P_α, which is dependent on the average queue length. The dropping probability P_α varies linearly from 0 to a maximum dropping probability of P_{max} and is computed as

$$P_a = \frac{P_b}{1 - N_b P_b},$$

where N_b is the number of packets since the last marked packet and P_b is the marking probability (i.e., to mark packets eligible for dropping), given by

$$P_b = P_{b\max} \frac{L_{av} - \Theta_{min}}{\Theta_{max} - \Theta_{min}},$$

where $P_{b\max}$ is the maximum possible marking probability, Θ_{max} and Θ_{min} are the maximum and minimum thresholds, respectively, and L_{av} is the average queue size that lies between Θ_{max} and Θ_{min} as shown in Figure 9. If the average queue length exceeds Θ_{max}, the packet is discarded unconditionally. The purpose of using the average instead of the instantaneous queue size is to filter out transient congestion at the node router.

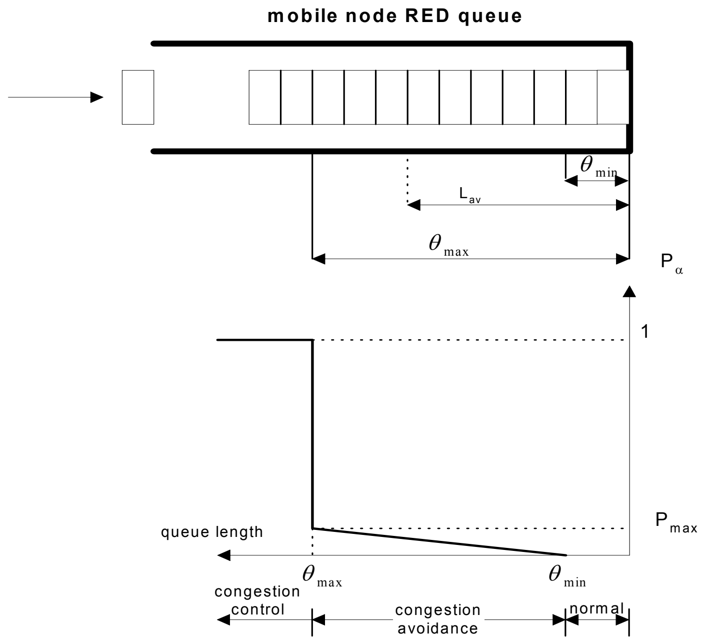


Figure 9. RED queue management scheme (From [14]).

C. SUMMARY

This chapter discussed QoS issues with emphasis on real-time traffic. Due to its distinct characteristics, real-time traffic requires preferential treatment in terms of both protocol structure and QoS guarantees. The chapter also described the RED algorithm. The QoS schemes RED and FIFO are used for the performance comparison of the proposed algorithm, presented in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SELECTIVE EARLY DISCARD (SED)

In this chapter we present a new active queue management scheme called the Selective Early Discard (SED) algorithm for real-time traffic over IP based networks. The key idea behind SED is simple: the more the errors are spread, the better the performance. This is not a new idea as it has been implemented in wireless digital channels in the form of interleaving, but it is a new idea for implementation in the management of the packet discarding policy algorithms. The reason for implementing interleaving in wireless channels is to reduce burst errors. Similarly, the problem with the existing queuing schemes for buffer management in the case of real-time traffic is that they suffer from buffer overflows, leading to significant degradation in performance.

The proposed algorithm takes into consideration the specific characteristics of the real-time traffic and provides a solution that keeps all the advantages of the existing active queue management QoS schemes and reduces as much as possible burst errors due to buffer overflow. The proposed queuing scheme is able to provide acceptable QoS guarantees by adjusting the queue parameters according to the traffic in environments in which the existing algorithms do not work satisfactorily.

SED's performance evaluation scenario considers only real-time traffic. Although a majority of today's network traffic is mixed (non real-time and real-time), this real-time traffic only scenario is used in order for the results to clearly reflect the performance benefits of SED for real-time traffic compared to other existing algorithms. In the current literature, new QoS service architectures have been proposed for IP-based networks that classify the traffic into multiple classes using MultiProtocol Label Switching (MPLS). In these service architectures, each service class implements a QoS scheme like FIFO or RED [48, 49]. Therefore, having only real-time traffic in the performance evaluation of the SED algorithm is not a simplification but rather a valid assumption for such architectures.

A. SED ALGORITHM

The main objectives of the Selective Early Discard (SED) algorithm are congestion avoidance by selectively discarding packets and minimizing the burst errors due to buffer overflow by spreading them as much as possible.

SED uses two thresholds. When the length of the queue exceeds these thresholds, we discard specific packets in order to spread the error. Considering that we have real-time traffic, which means that we can neither retransmit the discarded packets nor reduce the rate of the sources, SED spreads the packet loss as much as possible. SED controls the packet delay by adjusting the position of the thresholds from the head of the queue of finite size.

Pseudocode in Figure 10 outlines the SED algorithm. SED has two thresholds, Θ_1 and Θ_2, and three parameters: selective dropping 1 (SD1), SD2 and SD3. The parameter L is the instantaneous queue size and K is the maximum queue size. There are three congestion states. In the first state, for queue length less than Θ_1, no packets are dropped. If the buffer occupancy exceeds Θ_1, we drop the first packet in the queue that has a sequence number that is an integer multiple of SD1. If there is no packet that satisfies this relation, no packets are dropped. For queue size above Θ_2, we drop the first packet in the queue with a sequence number that is a multiple of SD2. If no packet in the queue satisfies this relation, again we do not drop any packet. Finally, if the queue size exceeds the buffer capacity, we discard the packet with sequence number that is a multiple of SD3 or the first packet in the queue.

B. PACKET DROPPING STUDY

SED's main goal is to drop packets for congestion avoidance in the presence of real-time traffic. We now study the effects of packet dropping on the voice quality in networks supporting voice over IP (VoIP) services.

```
For each arrival packet
        Enqueue packet
        Calculate the queue size, L
  if { $\Theta_1 < L < \Theta_2$ }
      if { there is a packet in the queue
            that satisfies
            [(sequence number) % SD1 = 0] }
            drop this packet
  else if { $\Theta_2 < L < K$ }
      if {there is a packet in the queue
            that satisfies
            [(sequence number) % SD2 = 0] }
            drop this packet
  else if { $L > K$ }
      if {there is a packet in the queue
            that satisfies
            [(sequence number) % SD3 = 0] }
            drop this packet
      else {drop the first packet in the queue}
```

Figure 10. Pseudocode for the SED algorithm.

1. Voice Quality and Packet Loss

Voice quality can be measured in terms of a dimensionless quantity called impairment factor, I [39]. Table 1 lists numerical values of I and the corresponding perceptual voice quality. The voice impairment factor can be expressed as

$$I = I_d + I_e - A \quad (4.1)$$

where I_d represents voice impairment due to long one-way transmission times (delay), I_e accounts for impairments caused by equipment and system related factors, and A is called expectation factor that depends on the network access method used [39].

The G.113 standard enumerates I_d as presented in Table 2. The standard also specifies I_e for certain coders: 0 for G.711 and 10 for G.729 and G.729a. These I_e values do not take packet losses into account. Measured voice quality at several discrete packet loss levels for two voice coders used in VoIP are reported in the literature as listed in Table 3 [34]. When packet losses occur, measured I_e values in Table 3 are recommended instead of those in G.113. Table 4 lists expectation factor A for typical voice networks [39].

I	Voice Quality
5	Very good
10	Good
20	Adequate
30	Limiting case
45	Exceptionally limiting case
55	Users likely to complain strongly

Table 1. Voice impairment values and the corresponding perceptual voice quality (After Ref. [39]).

Delay (msec)	150	200	250	300	400	500	600	800
I_d	0	3	10	15	25	30	35	40

Table 2. Voice impairment values due to transmission delay I_d (After Ref. [39])

Packet Loss (%)	0	1	2	3	4	5	6	7	8	9
I_e for G.711	0	8	12	18	22	26	28	30	32	34
I_e for G.729/G.729a	10	15	20	25	30	34	38	40	42	44

Table 3. Measured I_e values (After Ref. [34])

Voice Network Access Method	Expectation factor (A)
Conventional telephone line	0
Local area wireless network (cordless phone)	5
Wide area wireless network (cell phone)	10
Satellite	20

Table 4. Expectation factor A for typical voice networks (After Ref. [39]).

Consider a G.729 coder used in a cell phone with a 4% packet loss and 150 ms delay. From Tables 2 and 3, we can determine $I = I_d + I_e - A = 0 + 30 - 10 = 20$. Referring to Table 1, this value indicates *adequate* voice quality. For a packet loss of 7%, keeping the other parameters the same, the voice quality degrades to I = 30 (*limiting case* as in Table 1) which means that the conversation is understandable with moderate effort. For satellite channels, assuming a delay of 250 ms and a packet loss of 7%, the voice quality is acceptable (I = 30, *limiting case*).

Voice quality due to packet losses can be measured in terms of the widely used Mean Opinion Score (MOS) on a scale ranging from excellent (5) to bad (1). Figure 11 shows plots of MOS as a function of packet losses for G.729 and G.723.1 [33]. Results indicate that as the packet loss increases, the voice quality degrades. However, even at an error rate of 5%, a MOS score of approximately 2.9 was achieved, which corresponds to acceptable voice quality.

Error concealment techniques can be used to mitigate the effects of packet losses. Packet insertion for error concealment of the discarded packets is used in VoIP for the purpose of maintaining the timing relationship in a stream of packets. In silence substitution, a blank packet is substituted for the duration of the lost packet. In the packet repetition approach, a discarded packet is replaced by a copy of the packet immediately preceding the discarded packet. Two other techniques, interpolation and regeneration, are proposed in the current literature for concealing discarded packets. These are computationally more complex and relative to the insertion-based methods, the improvement achieved by these schemes is marginal, at best [35]. To provide toll quality in VoIP schemes, a packet loss of less than 2% and use of insertion based error concealment is recommended [33].

Consecutive packet losses significantly affect the voice quality. The effect of a missing packet on the listener is dependent on the size of the packet. Typically voice packets are 20 to 30-ms long. The smallest meaningful element of speech, the phoneme, has an average size of 80-100 ms. That means a loss of one packet generally does not adversely affect the voice intelligibility. In contrast, in the case of consecutive packet losses, 40 to 60 ms of speech may be missing, which may cause considerable degradation in voice quality. A whole phoneme could possibly be missed, and the substitution error concealment methods do not work well. Silence substitution results in gaps while packet repetition results in harmonic artifacts or beeps. In summary, consecutive packet losses cause significant degradation in voice quality [33], [34], [39].

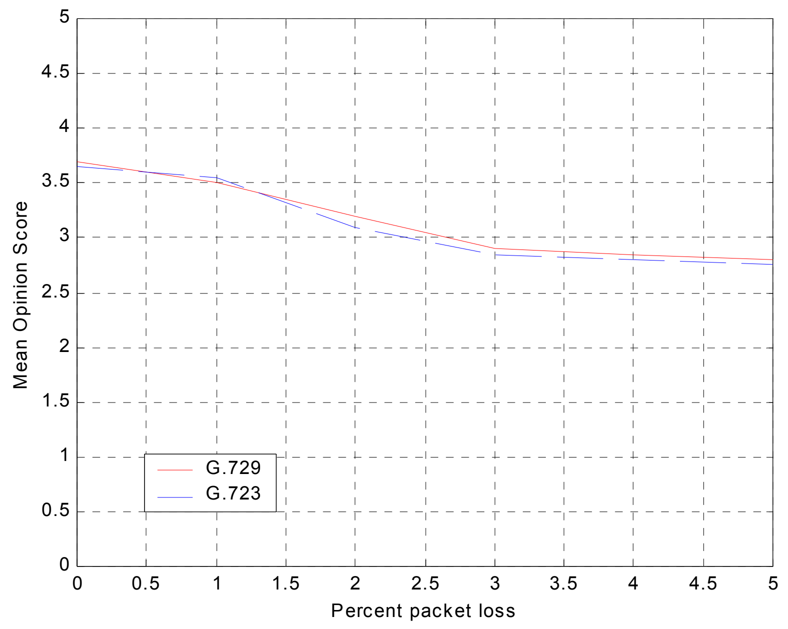


Figure 11. Packet Loss effects for G.229 and G.223 speech coders [After Ref. 33].

2. Experimental Packet Dropping Study for FS1016

This section presents experimental results using a Federal Standard 1016 (FS1016) coder with packet losses. FS1016 is based on the Code Excited Linear Predictive (CELP) coding algorithm. In FS1016, voice waveforms are first sampled at 8 kHz with a precision of 16 bits per sample. The CELP compressed voice packets represent 30-ms segments of voice or 240 samples and contain 144 bits. For these experiments, these packets are then subjected to loss using a Matlab function. The voice packets after losses are decompressed and played back.

For the purposes of designing an effective packet dropping algorithm, the objective is to figure out a packet discard pattern that provides the lowest voice quality degradation for the same amount of packet loss. This study defines quality according to three levels of perception: small degradation, significant voice degradation and unacceptable voice quality.

Table 5 summarizes experimental results of several speech files and error patterns. Three different error patterns were investigated in this experiment. First, the errors are spread as much as possible. For example, in the case of 5% error, we uniformly discard one packet every 20. Second, packets are discarded at random. For example, for 5% error, we lose 5 packets randomly in every 100 packets. Third, two consecutive packets are discarded. For example, with 5% error, we discard the 19th and 20th packets every 40 packets. In all cases we use silence substitution in place of a lost packet. The results reported in Table 5 are solely based on the author's perception of reconstructed speech after packet loss.

Results reported in Table 5 are based on the author's judgment of voice quality for different packet loss patterns. Table 5 indicates that the quality of voice is best when the errors are spread as much as possible. The consecutive packet errors cause a significant degradation in voice quality, which was found to be unacceptable if the error is more than 5%.

Percentage of packets discarded	**Voice quality**		
	Spreading the error	**Random error**	**Error in 2 consecutive packets**
5%	Small	Small	Significant
10%	Small	Significant	Unacceptable
15%	Significant	Unacceptable	Unacceptable

Table 5. Reproduced voice quality as observed by the author for three error patterns.

C. SED/IO ALGORITHM

Selective Early Discard with IN/OUT (SED/IO) is an extension of the SED algorithm that uses traffic prioritization in order to provide better service to specific flows. The traffic prioritization is based on DiffServ [27] in which packets are marked as either IN or OUT. Also, each type of traffic (IN or OUT) uses a separate SED algorithm with a different set of parameters for more flexibility in providing the desired QoS.

Figure 12 provides the pseudocode of the extended algorithm. For each packet arrival, the total queue size is calculated and the packet's differentiated field is examined. If it is an IN packet, the SED algorithm for IN packets is implemented; otherwise, the SED algorithm for OUT packets is implemented. Packets are discarded according to the specific parameters of each SED algorithm. Additionally the SED algorithm for OUT packets typically has a third threshold lower than the maximum buffer capacity. That means if the queue size exceeds the third threshold for OUT packets and an OUT packet has arrived, an OUT packet with sequence number that is an integer multiple of SD3 or the first packet in the queue is discarded.

```
For each arrival packet
  Enqueue packet
  Calculate the queue size, L
    if { it is an IN packet }
        implement algorithm
        for IN packets
    else
        implement algorithm
        for OUT packets
```

Figure 12. The SED/IO algorithm.

By adjusting the parameters for the IN and OUT packets, we are able to preferentially treat the high priority (IN) packets in two ways. First, using IN thresholds larger than the corresponding OUT thresholds, which results in early dropping of OUT packets with respect to IN packets. Second, by choosing lower dropping parameters (SDs) for OUT packets, we increase the dropping probability of OUT packets.

D. SED AND SED/IO USING TIMESTAMPS

QoS implementation in IP based networks takes place in the network layer in which routers read the IP header and apply pre-specified rules for each packet. However, in the case of real-time traffic, the RTP header contains significant information in the timestamp field. The timestamp contains the creation time of each packet; therefore, a packet delayed beyond a bound can be detected in an intermediate router. By dropping this delayed packet, the queue size does not increase, hence the queuing delay for packets arriving after the dropped packet is decreased. Examination of the RTP header by a router presents some difficulties but it is feasible as explained in the next section.

SED with timestamps spreads the error as much as possible and reduces the queuing delay. Pseudocode of the algorithm using timestamps is provided in Figure 13. At each packet arrival, the timestamp in the RTP header is read and compared with the local time in the router, and the packet delay is computed. A packet is considered late whenever its measured delay exceeds the maximum allowed predetermined one-way delay (D_{max}) and, if such is the case, it is dropped. If the packet is not late, it is further processed by SED or SED/IO algorithm as explained in the previous sections.

```
For each arrival packet
  Enqueue packet
  Read the timestamp in RTP header
    if { (pkt time – Local Time) > $D_{max}$ }
        drop the packet
    else
        implement SED or SED/IO algorithm
```

Figure 13. Pseudocode of the SED and SED/IO algorithms using timestamps.

E. IMPLEMENTATION ISSUES

In this section, we describe how the SED and SED/RIO can be implemented in IP-based networks. The issues we address are marking of packets, dropping of a chosen packet, and reading of the RTP header.

The packets can be marked using bits from the Differentiated Services (DS) byte field. This field is defined in DiffServ [27] and is intended to supersede the existing definitions of the IPv4 TOS octet [30] and the Ipv6 Traffic Class octet [31]. The reconstructed field is presented in Figure 14 and has the following subfields: the 6-bit differentiated services codepoint field (DSCP) and the 2-bit Currently Unused (CU) field. The DSCP is further divided into a 1-bit "IN" field, which indicates whether the packet conforms to a predefined profile with respect to the traffic policies at a network boundary, and a 5-bit PHB field, which marks the required per-hop-behavior of each packet. The two bits in the CU field are ignored by DiffServ nodes when determining the per-hop-behavior to apply to a received packet.

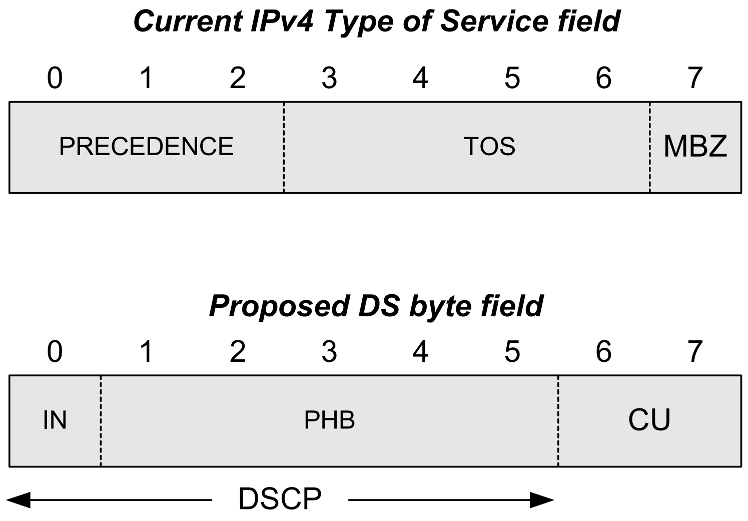


Figure 14. Allocation of bits in the type of service byte (Ipv4) to support DiffServ [After Ref. 32].

The SED/RIO algorithm uses the six bits of the DSCP field to preferentially treat certain flows as in DiffServ. Additionally, both SED and SED/IO utilize the two unused bits of the CU field to indicate whether a packet is droppable and in which congestion state. More specifically, each packet is marked at the source using the CU bits as follows: "00" for a packet that is not droppable, "01" for a packet droppable in the first congestion state, "10" for the packets droppable in the second state, and "11" for the packets droppable in the third state. This classification of packets into multiple droppable states is based on the sequence numbers and the predetermined SD parameters. The sequence numbers, which are created at the source at the time of packet generation, are available from the corresponding field in the RTP header. When the IP header is added to a packet, the sequence number is examined and the droppable state of the packet is marked using the CU bits in the DS byte field. For example, if SD1 for a specific source is 20, all packets generated by this source having sequence numbers that are integer multiples of 20 are droppable in the first congestion state. Therefore, these packets are marked in the CU field with “01,” indicating their droppable state. This allows the routers with DiffServ capabilities to read the two CU bits in the DS byte field and to acquire the necessary information for implementing the SED algorithm.

There are three important considerations in the implementation of the SED algorithm and its extensions. First, SED and SED/IO take advantage of the sequence numbers in order to spread the error. Second, traffic is divided into four classes based on the droppable state to which each packet belongs. Thus, the amount of state information in each node is reduced to the number of droppable states rather than to the number of flows, leading to increased scalability. Third, the predetermined SD parameters can be different for different real-time applications (e.g., voice, video) because marking of packets takes place at the source. Therefore, SED is capable of discarding packets according to specific requirements of each application.

The next step in the implementation of SED is to examine the mechanism of dropping a packet. In order to avoid extensive computations for each incoming packet, the following procedure is proposed. Consider a 3×K array, where K is the maximum queue size in terms of packets and 3 indicates three droppable states; each row corresponds to one droppable state. When a packet arrives, the router examines its CU field to see if it is droppable. For example, if a packet is droppable in the second state (CU field of “10”) the position from the head of the queue for this packet is stored in the second row of the matrix. Entries in a row indicate the droppable packets in the corresponding state. When the queue size exceeds a threshold, the droppable packets within the queue for this congestion state are available from the corresponding row. For example, if the queue is in the second droppable state $\{\Theta_2 < L < K\}$, the first packet from the second row of the matrix is dropped. If the second row is empty, then no packet is dropped. Each time a packet is sent or the first packet in the queue is dropped, the positions from the head of the queue for the droppable packets in the matrix are updated. Additionally, when a packet is dropped from a droppable state the index elements of the matrix are updated for all the packets that are farther from the head of the queue. Figure 15 illustrates an example of updating the matrix elements when a packet belonging to a droppable state is dropped. Assuming that a packet arrives and the queue size is in the second droppable state ($\Theta_2 < L < K$), the packet corresponding to the first index element of the second row (45th packet of the head of the queue) is dropped. Then, the index elements in the matrix are decreased by one if their value is greater than the value corresponding to the dropped packet (45).

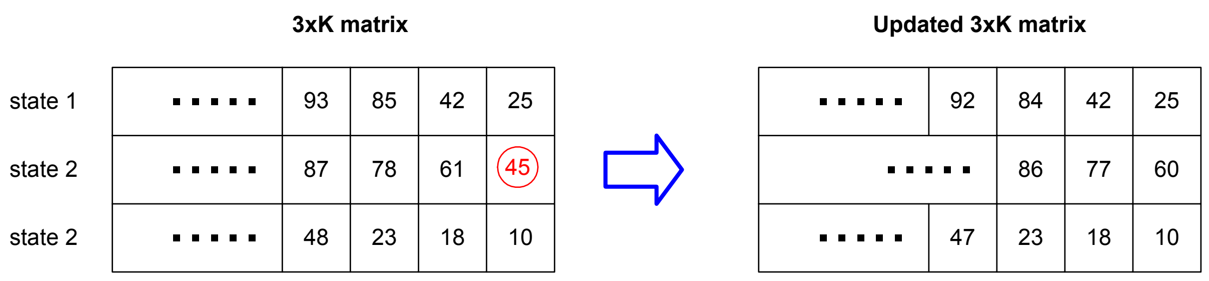


Figure 15. Example of updating matrix values in case of discarding the packet of the second droppable state.

In the above implementation, no extensive computations are required in the time-critical packet-forwarding path in order to drop packets. Updating the positions of the droppable packets from the head of the queue can be performed in parallel with packet forwarding. As it can be performed as a low priority task, the node's ability to process packets is not affected.

F. SUMMARY

This chapter presented a new packet dropping algorithm called Selective Early Discard (SED). An experiment involving packet loss in voice traffic is conducted to study the effects of loss on voice quality for different loss patterns. SED was extended to SED/IO to provide priority to critical data and to SED with timestamps to selectively discard overly delayed packets at the intermediate nodes. Also, implementation issues for SED and its extensions were addressed.

THIS PAGE INTENTIONALLY LEFT BLANK

V. BIT DROPPING

This chapter investigates an active queue management technique that drops bits instead of whole packets. This method is motivated by work in [36] in which a bit-dropping scheme for ADPCM voice packets is presented. A few different approaches for dropping bits and recovering the missing bits are discussed. Then, a bit-dropping study for the FS 1016 CELP codec is examined for its tolerance to bit dropping with respect to voice quality and a proposed dropping pattern for this coder is given.

A. BIT DROPPING FOR MULTIMEDIA TRAFFIC

Compressed multimedia data is formed into frames composed of the parameters and coefficients associated with the particular compression algorithm used. The key idea in bit dropping is to selectively drop bits within a buffer during times of congestion. This loss of bits will result in signal degradation, but many multimedia applications (such as voice) are loss tolerant so that an acceptable signal can be recovered. A disadvantage of a bit-dropping scheme is that in order to selectively drop bits, the algorithm must have access to the data payload. This requires a modification to the router's functionality. The determination of when to apply bit dropping can be based on buffer thresholds, such as those used with SED: when the buffer gets too full, bit dropping would be implemented.

Bit-dropping techniques can be divided into two general categories: dropping of less significant bits (of coefficient values for example); and dropping of certain segments of each frame (such as dropping an entire coefficient or parameter value). Dropping of less significant bits can be done similar to the one presented in [36], where each voice packet is organized at the source into four blocks: the first block contains the least significant bits, while the fourth block contains the most significant bits. During periods of congestion, one or more blocks are dropped (less significant block first) in order to speed the packet service time within the queue.

On the other hand, in compressed voice, specific parts from each voice packet can be dropped and substituted by using previous packet substitution or intraframe substitution. Substitution using the previous packet means that the corresponding bits from the previous received packet replace missing bits. Intraframe substitution is the replacement of the missing bits from similar bits within the same frame. As in the dropping of less significant bits, voice packets are formed into blocks and specific blocks are dropped during periods of congestion in the buffer, based on their sensitivity on voice quality.

Based on the above discussion, a pseudocode of a bit-dropping algorithm can be as depicted in Figure 16. There are two thresholds Θ_1, Θ_2 and the maximum buffer capacity is K. The instantaneous queue size is L. For each incoming packet, the queue length is calculated and specific blocks are dropped when queue size exceeds a threshold. For example, if it exceeds the first threshold the least significant block of the just arrived packet is dropped. Note that when a block is dropped, the IP length field must be updated. Finally, if the queue size exceeds the maximum buffer capacity the packet is dropped unconditionally.

```
For each arrival packet
        Enqueue packet
        Calculate the queue size, L
  if { $\Theta_1 < L < \Theta_2$ }
        (drop the least significant block
        of the last packet)
        (update IP's length field)
  else if {$\Theta_2 < L < K$ }
        (drop the second least significant block
        of the last packet)
        (update IP's length field)
  else
        (drop the last packet)
```

Figure 16. Pseudocode of the bit-dropping scheme.

B. BIT DROPPING

In this section, a study is conducted using a CELP coder for its tolerance to bit-dropping. A CELP coder constructs a speech frame based on codebooks. That means an error in a bit, even if it is a less significant bit, can significantly degrade the voice quality. Additionally, each frame has adaptive and stochastic indices and gains that depend on the voice characteristics. Since voice characteristics do not change much over a short period of time, substituting indices and/or gains is possible. This study aims to provide general guidelines about the degradation in voice quality when specific bits of voice frames are dropped. Note that results are mainly based on listening tests using the author's perception of reconstructed speech after bit-dropping.

1. Experiments Using the FS1016 CELP Coder

The particular implementation being considered for CELP encoded speech is the 4800-bps Federal Standard 1016 (FS1016) [37]. Table 6 lists the main characteristics of the codec. FS1016 divides the speech to be coded into 30-ms frames, each of which is further divided into four 7.5-ms subframes. Consequently, each frame contains 240 speech samples at a sampling rate of 8,000 sps. This codec uses a Hamming parity code (15,11) for forward error correction. The protected bits are the three most significant bits of the first and third subframe's adaptive indexes, and the most significant bit of the adaptive gains and the expansion bit. The frame synchronization bit alternates between zero and one from frame to frame.

	Linear Predictor	**Adaptive codebook**	**Stochastic codebook**
Update	30 ms	30/4=7.5 ms	30/4=7.5 ms
Parameters	10 LSPs	256 codewords	512 codewords
Bits per frame	34 (3,4,4,4,4,3,3,3,3,3)	Index:8+6+8+6 Gain: 5x4	Index: 9x4 Gain: 5x4
Rate (4800 bps)	1113.33	1600	1866.67
	* The remaining 200bps are used as follows: 1 bit per frame for synchronization, 4 bpf for FEC and 1 bpf to provide future expansion(s) of the coder.		

Table 6. Federal Standard 1016 characteristics (After Ref. [37]).

a. Dropping of Less Significant Bits

In the experiments, FS1016 CELP v.3.2 for Matlab [38] was used, having as input Windows standard 16 bit .wav files sampled at 8,000 Hz. This code was modified in order to provide results for different bit dropping patterns. In particular, the simulation model inserts zeros into specific bits in each packet according to a bit-dropping scenario. Then the packets are played out in the decoder assuming that the dropped bits are not transmitted.

The bit dropping schemes shown in Table 7 are the least significant bits for each index/gain. For example in the dropping scheme of 10 bits, the least significant bit of the adaptive gain of each second and fourth subframe, all stochastic indexes and gains is dropped. According to the listening tests of these bit-dropping schemes, only in the case of dropping 10 bits is the voice quality acceptable.

Bits dropped	**LSP pairs (10 pairs per frame)**	**Adaptive (4 per frame)**		**Stochastic (4 per frame)**	
		Gain	**Index**	**Gain**	**Index**
10	0+0+0+0+0+0+0+0+0+0	0+1+0+1	0+0+0+0	1+1+1+1	1+1+1+1
21	0+1+0+1+0+1+0+1+0+1	1+1+1+1	1+1+1+1	1+1+1+1	1+1+1+1
36	1+2+2+2+2+1+1+1+1+1	1+1+1+1	1+1+1+1	1+1+1+1	1+1+1+1

Table 7. Bit dropping schemes.

Due to complexity in the implementation of a bit dropping technique and the small bandwidth savings in the 10-bit dropping scheme, the use of this technique is not recommended. However, by implementing a bit-dropping scheme based on bit sensitivity instead of less significant bits, the bandwidth savings can be greater. This is the result of CELP encoded speech in which each bit has a different weight that influences the overall performance differently.

b. Dropping of LSPs, Indexes and/or Gains

This section discusses dropping of Line Spectrum Pairs (LSPs), and/or adaptive and stochastic indexes and gains to satisfy the objective of reducing the required bandwidth for transmission of voice. In the experiments, the original FS 1016 Matlab code was modified using the necessary Matlab functions for extracting and processing each packet. Substitution using the previous packet and intraframe substitution as mentioned in a previous section are examined in order to recover the missing bits.

In the simulation scenario of substitution using the previous packet, the missing components are changed every other packet in order to avoid the propagation of error. For example, in the case of transmitting only 8 LSP pairs instead of 10, in one packet LSP pairs 8 and 10 are dropped while in the consecutive packet LSP 7 and 9 are not transmitted.

Based on listening tests, intraframe substitution provides better voice quality compared to previous frame substitution. This was expected because, in FS1016, a speech frame is 30 ms long and thus, the characteristics of the speech signal do not change much within each frame. In contrast, from frame to frame, the change in the values of LSPs, indices and gains can typically be significant.

Using intraframe substitution, the behavior of LSP pairs, adaptive and stochastic indexes and gains is examined separately by defining quality according to four levels of perception: slight degradation, small degradation, significant voice degradation and unacceptable voice quality.

Results indicate that by transmitting only 7 of the 10 LSP pairs, the voice degradation is small. The total bandwidth savings per frame is 10/144 = 7%. Furthermore, slight voice degradation exists when missing one LSP and, in contrast, significant degradation exists when transmitting 6 LSPs. Finally, when transmitting only half of the LSPs, the voice quality is unacceptable.

Adaptive and stochastic indices and gains are also examined for an indication of possible bandwidth savings. Results are summarized in Table 8. In general, adaptive indices and gains are more sensitive compared to stochastic indices and gains. For example, when transmitting half of the adaptive indices, the quality is unacceptable. In contrast, when transmitting half of the stochastic indices or gains, only small voice degradation occurs. Additionally, when only one stochastic index or gain is kept per frame, significant degradation occurs but voice quality is still acceptable.

	Number of Dropping indexes per frame		
	1	**2**	**3**
Adaptive indexes	Significant	Unacceptable	---
Adaptive gains	Small	Small	Unacceptable
Stochastic indexes	Slight	Small	Significant
Stochastic gains	Slight	Small	Significant

Table 8. Results for dropping adaptive and stochastic indexes and gains.

c. *Bit-Dropping Scheme for FS1016 CELP Coder*

Based on the above results, dropping of less significant bits does not provide large bandwidth savings with respect to voice quality, intraframe substitution is better when compared to previous frame subsitution, and adaptive indexes and gains are more sensitive.

The dropping of less significant bits does not provide good performance because each bit in a compressed voice packet carries more information and, therefore, is more sensitive to loss than that in an uncompressed voice packet. The intraframe substitution of LSPs, indexes and gains in CELP coders performs better than the previous frame substitution due to the distinct characteristics of CELP coders. Specifically, in FS1016, a speech frame is 30-ms long and thus, the characteristics of the speech signal do not change much within each frame. Therefore, the values of the indices and gains in consecutive 7.5-ms subframes change less compared to subframes from previous packets. Finally, results based on listening tests indicate that the loss of stochastic indexes and gains is better than the lack of the adaptive indexes and gains.

By taking these observations into account, the proposed bit dropping scheme does not transmit the stochastic index and gain of the second subframe. Furthermore, the 7 less significant bits from the stochastic index of the fourth subframe and the 10th LSP are not transmitted. In order to conceal the signal loss due to the missing bits, intraframe substitution is used. More specifically, the stochastic index and gain of each second subframe are substituted from the corresponding index and gain from the first subframe. Also, the missing bits of the stochastic index of the fourth subframe are substituted by bits from the third subframe. Finally, the 9th LSP substitutes for the missing LSP. Overall, the proposed scheme transmits only 120 bits of each frame, leading to a 16.6% bandwidth saving. Listening results show that by using this scheme the degradation of voice is acceptable. Additionally, the proposed scheme is superior compared to a similar one that drops the least significant bits in terms of bandwidths. In particular, the scheme that drops 24 least significant bits results in unacceptable voice quality.

C. SUMMARY

In this chapter a bit-dropping scheme utilizing different techniques were discussed. Additionally, a bit-dropping study for a specific CELP coder, the FS1016, was conducted. Results, based mainly on listening tests, showed that speech frames can be transmitted using fewer bits with tolerable performance degradation. The proposed bit-dropping scheme in which only 120 of the 144 bits were transmitted for each frame (16.6% bandwidth saving) provides acceptable voice quality. This scheme can be used when congestion occurs in the buffer in order to avoid buffer overflow.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS AND DISCUSSION

MANETs are characterized by dynamic multi-hop topologies in which the nodes can move arbitrarily, changing the topology rapidly depending on the scenario. This results in traffic flows and service times that do not necessarily follow a Poisson or exponential distribution. Additionally, partitioning/merging of traffic and non-independent service distributions make a mathematical analysis based on queuing theory infeasible [45]. Consequently, the performance evaluation in this thesis is based on simulations. This chapter presents the simulation environment and simulation results for the evaluation of the proposed QoS algorithms.

A. SIMULATION ENVIRONMENT

The simulation software used for evaluation of the proposed QoS schemes was the Network Simulator 2 (NS2) version 2.1b6 [42] running on a Linux RedHat 6.2 platform. At the time of writing this thesis, NS2 seems to be the standard tool to simulate ad hoc networks. Many routing protocols that are used in MANETs are available in NS2. This section describes the basics of NS2 and the modifications to some NS2 functions to simulate the QoS algorithm reported in this thesis. Additionally, the simulation parameters and the performance metrics are discussed.

1. Network Simulator 2

The NS2 was developed by the network research group at the Lawrence Berkeley National Laboratory (LBNL). Currently, NS2 is part of the Virtual InterNetwork Testbed (VINT) project, which is a collaborative effort between LBNL, University of California at Berkeley (UCB), the University of Southern California/Information Science Institute (USC/ISI) and Xerox Palo Alto Research Center (Xerox PARC). The goal of the VINT project is to extend the NS simulator so that network researchers can study the complex interactions between network protocols (e.g., unicast routing, multicast routing, TCP, reliable multicast, integrated services, etc.) in complex topologies with a rich set of traffic generators [42].

2. Simulation Overview

The NS2 is a discrete event simulator written in C++ programming language and uses Object Tool Command Language (OTcl) interpreter at the user level. Figure 17 depicts a simplified user's view of NS2 [42]. A typical simulation in NS2 includes several steps. First the user creates the OTcl scripts, which are the input files to the simulator. These files consist of a scenario file that describes the movement pattern of the nodes and a communication file that describes the traffic in the network. Then the simulator initiates an event scheduler and sets up the network topology using the network objects and the plumbing functions in the library. Also, it informs the traffic sources when to start and stop transmitting packets through the event scheduler. The result of this procedure is the generation of a trace file. The granularity of the trace files is determined prior to simulation in the OTcl scripts. Typically, the trace files are parsed using Perl or another Linux shell script allowing the performance metrics of interest to be obtained. Finally, the analyzed data from the trace files can be used for further manipulation and plot generation using other languages like Matlab. Another option that can be defined in the OTcl scripts is the visualization of the simulation run using the Network Animator (NAM).

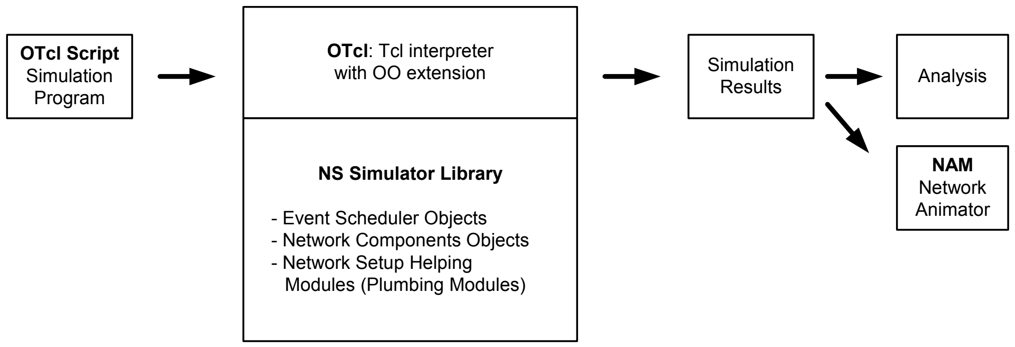

OTcl Script
Simulation Program
OTcl: Tcl interpreter with OO extension
NS Simulator Library
- Event Scheduler Objects
- Network Components Objects
- Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

Figure 17. A simplified user's view of NS2 (After Ref. [42]).

NS2 provides a wireless model consisting of mobile nodes that allow simulations of multihop ad hoc networks. Figure 18 shows the basic components of a mobile node. It consists of a routing agent, link layer (LL), Address Resolution Protocol (ARP), interface queue, MAC protocols, network interface, radio propagation model and a channel. A mobile node is responsible for forwarding packets to the destination. Packets can be received from an application or from another node. The routing agent decides the path of each packet to its destination, stamps it with this information, and sends the packet to the LL. If the ARP has the hardware address of the destination, it inserts the address in the MAC header. Otherwise, the packet is buffered and an ARP query is sent. Once the hardware address of a packet's next hop is known, the packet is sent to the interface queue. Next, when the MAC layer decides that it can send a packet onto the channel, it fetches the packet from the head of the queue and forwards it to the network interface. Finally, the packet is sent onto the radio channel. This packet is copied and delivered to all network interfaces at the time at which the first bit of the packet would begin arriving at the interface in a physical system. Each network interface stamps the packet with the receiving interfaces' properties and then invokes the propagation model.

3. Modifications

NS2 version 2.1b6 is an open platform that provides the necessary components for simulating multihop ad-hoc networks. However, for the evaluation of the proposed QoS schemes, changes had to be made. The main changes and added functions are related to the generation of script files, the implementation and integration of the QoS schemes, and the processing of the results.

Following the typical procedure in NS2, input script files were generated. For each simulation scenario, one Tcl script determines the traffic pattern and another the mobility scenarios of the nodes. A third Tcl script defines the characteristics of the mobile nodes, such as routing and queuing parameters, MAC protocol and physical layer parameters. Finally, the parameters that were traced during the simulation were defined in these files. Generally, the new Tcl scripts are similar to the ones available in NS2, with necessary changes for incorporation of the added functionality. Additionally, each simulation scenario uses its own unique Tcl input files.

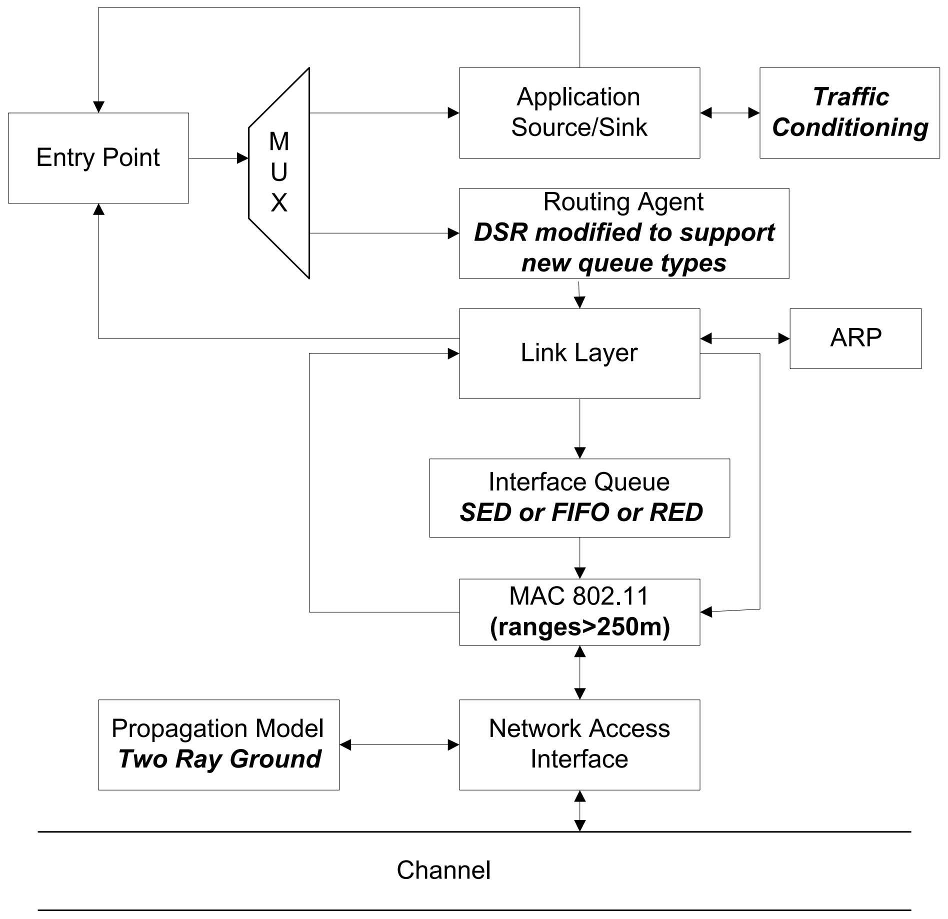


Figure 18. Schematic of a Mobile Node Protocol Stack in NS2 (After Ref. [42]).

Only one QoS scheme, FIFO, is offered in NS version 2.1b6 for implementation in MANETs. For the simulation of the RED algorithm, functions from other researchers were used [43], [44], [14]. The QoS schemes developed in this thesis were implemented in C++ and embedded into NS2. Appendix A contains the code of the proposed SED QoS schemes. Furthermore, the simulations include transmission range of 10 km for each node, which is larger than the default value of NS2 (250 meters). This expansion was made following the procedure in [14], which involves the modification of the MAC C++ code in order to change the standard timing parameters of the Distributing Coordination

Function (DCF). Also, the received and transmitted power of each mobile node's network interface was adjusted in accordance with the larger ranges.

The trace files were processed by parsing them into perl files for extracting of required data. Then, the data were further analyzed in Matlab and plotted.

4. Performance Metrics

Typically, the performance of a network is measured by using as metrics the average end-to-end delay, the packet loss and the throughout. All these metrics are used for the evaluation of the proposed algorithms. However, the traffic under investigation is real-time voice as the packet loss distribution significantly affects its quality; in particular, consecutive packet losses are of interest here.

Packet loss distribution can be measured as a function of the frequency of intervals between packet losses. The frequency of packet loss is calculated as

$$\text{Frequency of Interval } i = \frac{\text{Number of Occurrences of Interval } i}{\text{Total Number of Losses - 1}} \tag{6.1}$$

For example, if in 10 consecutive packets the 3rd, 4th, 5th, 7th, 9th and 10th packets are missing, we have 3 occurrences of losing 2 consecutive packets (interval equal to one) and 2 occurrences of losing packets separated by one packet (interval equal to two). As a result, considering that the total number of missing packets is 6, the frequency of one-packet interval is 3/(6-1) = 0.6 and of two-packet interval is 2/(6-1) = 0.4. In other words, the frequency value of the interval one represents the probability of losing one packet every other packet. For example, a frequency 0.1 for an interval of one means that we expect 10% of the total losses to happen in two consecutive packets. Burst losses are measured using the distribution of consecutive packet losses.

The performance metrics are mainly measured against mobility, offered traffic load and network size. Mobility corresponds to the movement of the ad hoc nodes. In the simulations, the pause time of each node is changed while the speed of the node during motion remains constant. Traffic load scenarios consist of low traffic load corresponding to packet losses in the range of 0.5-1.5% and heavy traffic load corresponding to packet losses above 3%. Network size is represented in terms of the number of nodes and the size of the area in which mobile nodes roam.

5. Network Configuration

Many different network configurations were used in the simulation runs. Table 9 shows the parameter values used in NS2 simulations. The number of nodes is 2 or 10 and source rates are from 240 kbps to 300 kbps. The voice traffic in the simulations is represented by ON/OFF sources; typical values used for the on/off periods are 0.42 sec and 0.58 sec, respectively. It is assumed that silence detection is employed, hence packets are generated only during talk-spurt periods. The overhead from the RTP, UDP and IP layers is 40 bytes, which can be decreased to 2-4 bytes using compression [23]. In the simulations, the packet length was set to 210 bytes and the node velocity is in the range of 0-20 m/s. Moreover, the maximum transmission range of 802.11 is increased to 10 km, and different mobility scenarios are examined in terms of pause times. Simulations are run for various geographical sizes of 5×5, 10×10 and 30×30 km^2. Also of factor are the processing time and the computer's hard disk space.

Parameter	Range of values	Units
Source rate	240-300	kbps
Packet size	210	bytes
Traffic type	VBR/UDP	---
Queuing schemes	SED, RED, FIFO	---
Buffer size	160-200	packets
Routing protocol	DSR	---
MAC protocol	802.11	---
Transmitter range	10	km
Area size	5×5, 10×10, 30×30	Km^2
Number of nodes	0 or 10	---
Node velocity	0-20	m/sec
Pause time	0, 200, 400, 600, 800, 1000	sec
Transmitted power	50	watts
Antenna heights	10	m
Bandwidth	2	Mbps

Table 9. Parameters used during NS2 simulations.

B. SENSITIVITY OF THRESHOLDS

The network topology used in simulations for examining the sensitivity of SED's thresholds is shown in Figure 19. It represents a simple ad hoc network scenario in which two nodes are moving inside an area of 5×5 km^2 with a constant velocity of 20 m/sec (pause time 0). Each node has five hosts. In one node, the hosts are sources that send data while in the other node the hosts are destinations of the sources. The data rate of each source is 300 kbps, and the maximum buffer size is 200 packets. The small area of 5×5 km^2 was chosen in order for the routing losses to be negligible compared to those due to buffer overflow. The buffer size was selected according to the delay limitations. The delay and network congestion are directly proportional to the buffer size and source rate, respectively. The results presented in the following are obtained by averaging results from five independent simulation runs. The total number of packets sent from the sources is approximately 365,000 packets per source in each simulation run.

Based on the results reported for the VoIP [33], [34], [39] and the experimental results for the FS1016, the selected dropping parameters SD1, SD2 and SD3 were assigned values of 20, 10 and 5, respectively. Using the above described simulation model, the appropriate values of the first and second thresholds in terms of the queue length were investigated. The third threshold is always equal to the maximum buffer size. Results are presented for three different choices of first and second threshold values (Θ_1=170, Θ_2=190), (140, 180) and (100, 180).

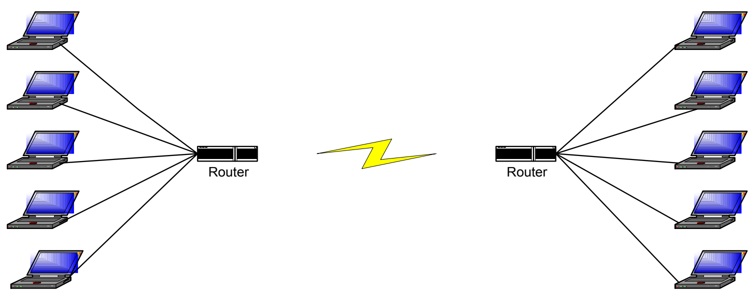


Figure 19. Network topology for investigation of thresholds for SED scheme.

Figure 20 shows the frequency of packet losses in terms of packet intervals. The packet losses are concentrated in intervals 20, 10 and 5, and this error pattern represents the scheduling of packet dropping according to parameters SD1, SD2 and SD3. The thresholds influence the frequency of losses in these intervals. For example, by selecting the threshold pair SED (100, 180), we have more packet losses in intervals of 20 while the threshold pair SED (170, 190) with larger values has more packet losses in intervals of 5.

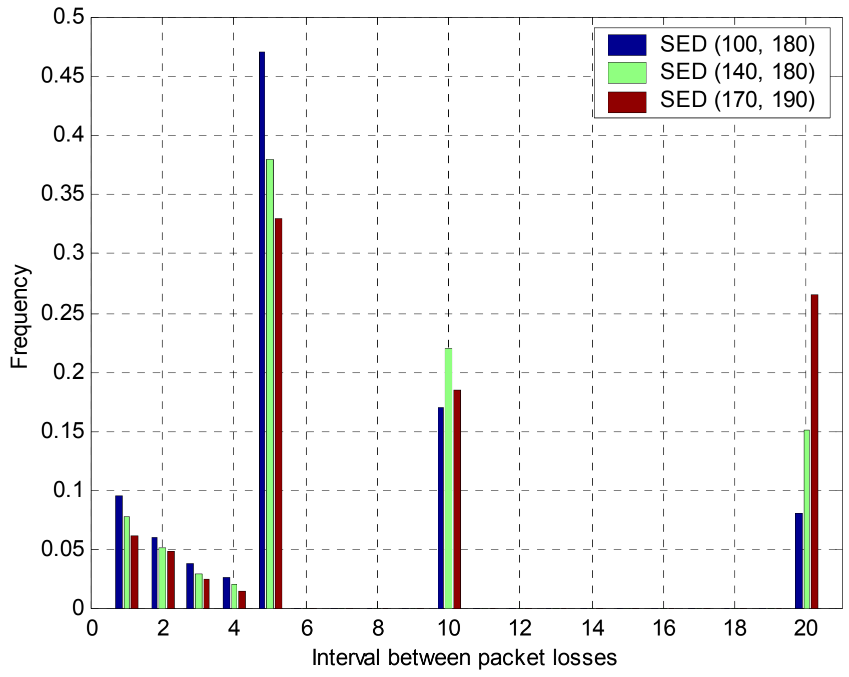


Figure 20. Frequency of packet loss in terms of packet intervals for the first 20 intervals. In each interval the first (left) bar corresponds to SED (170, 190), the second to SED (140, 180) and the third to SED (100, 180). Plots are obtained by averaging results from five connections.

By decreasing Θ_1, the packet errors are reduced in intervals 1-5, which simply means that burst errors are smoothed; the average delay is decreased, but more packets are dropped early. SED (170, 190) has an average delay of 149 ms and an error of 2.72% while has corresponding values for SED (100, 180) are of 137 ms and 3.22%, respectively. Therefore, there is a trade-off in the selection of the thresholds among the average delay, the total error, and better performance in the more sensitive area of packet loss. We choose SED (140, 180) in the remaining simulations as this pair is found to perform better than other choices.

C. COMPARISON WITH RED AND FIFO

In this section we compare the proposed algorithm with the RED and FIFO queuing schemes. The network configuration was similar to the one in the investigation of threshold values. Specifically, 2 nodes, area size of 5×5 km^2, pause time of 0 sec, source rate of 300 kbps and maximum buffer size of 200 packets were used. The thresholds for the SED algorithm were Θ_1=140 and Θ_2=180. The parameters of the RED algorithm were selected according to the guidelines in [29]. In particular, we set the queue weight parameter to 0.002, the maximum dropping probability P_{max} to 0.3, the minimum threshold to 160, and the maximum threshold to 200. It was observed that a larger Θ_{min} makes RED behave like FIFO while a lower minimum threshold results in significant increase in the total error compared to the error in the FIFO scheme.

From Table 10, the FIFO queuing scheme provides the smallest possible error 2.53% because it does not drop until the queue is full and has an average delay of 158 ms. SED does not cause a significant increase in the total error as RED does because SED drops only a small number of packets below Θ_2. The average number of droppable packets when the queue exceeds the minimum threshold for the first time is Θ_1/SD1. When Θ_2 is exceeded, more packets are dropped; however, by setting Θ_2 close to the max buffer capacity, the total error is kept low.

Queuing scheme	Loss Rate (%)	Average delay (ms)
FIFO	2.537	158
RED	3.096	142
SED 140-180	2.774	142

Table 10. Packet loss and average end-to-end delay.

Figure 21 presents the frequency of packet losses for the three queuing schemes. The SED algorithm provides much better performance in terms of spreading the error.

Considering the total error for the three algorithms (see Table 10), the RED and FIFO schemes cannot decrease the burst errors due to buffer overflow. From Figure 19, when the queue length is between the Θ_{min} and Θ_{max} thresholds, RED spreads the error more than FIFO due to the random drops.

Figure 22 shows the distribution of the consecutive packet losses. Clearly, consecutive packet losses are predominant for FIFO and RED (approximately 65% of the total losses) while SED has only 17% consecutive packet losses. Taking into consideration that voice degradation comes mainly from consecutive packet losses and/or losses in intervals of 2-4 packets, it is clear that SED provides better performance compared to FIFO and RED by spreading the error.

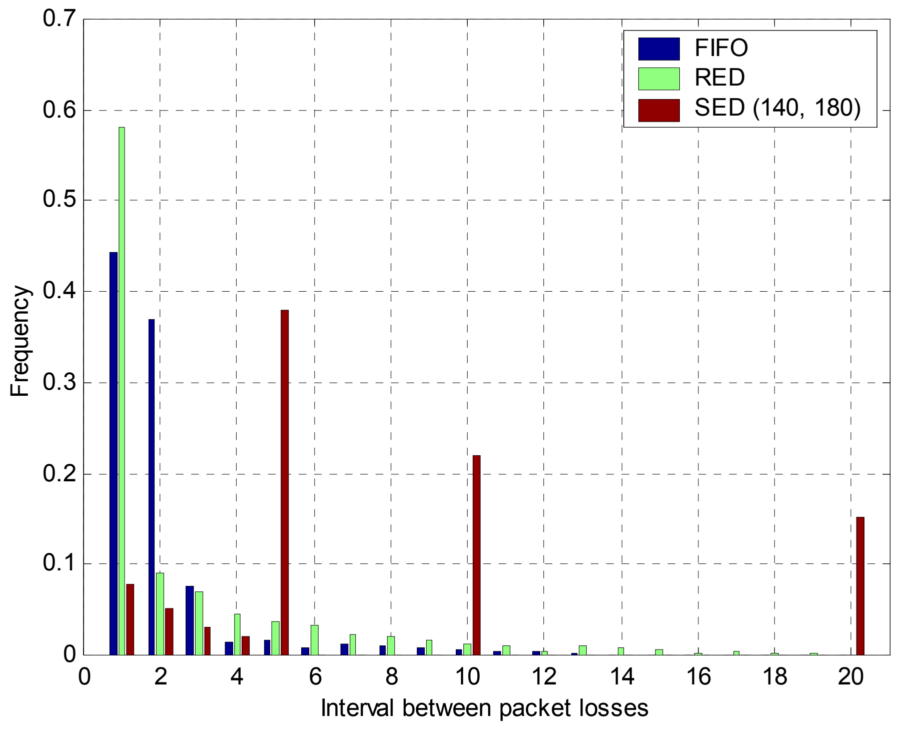


Figure 21. Frequency of packet loss in terms of packet intervals for the first 20 intervals. In each interval the first (left) bar corresponds to FIFO, the second to RED and the third to SED (140, 180). Plots are obtained by averaging results from five connections.

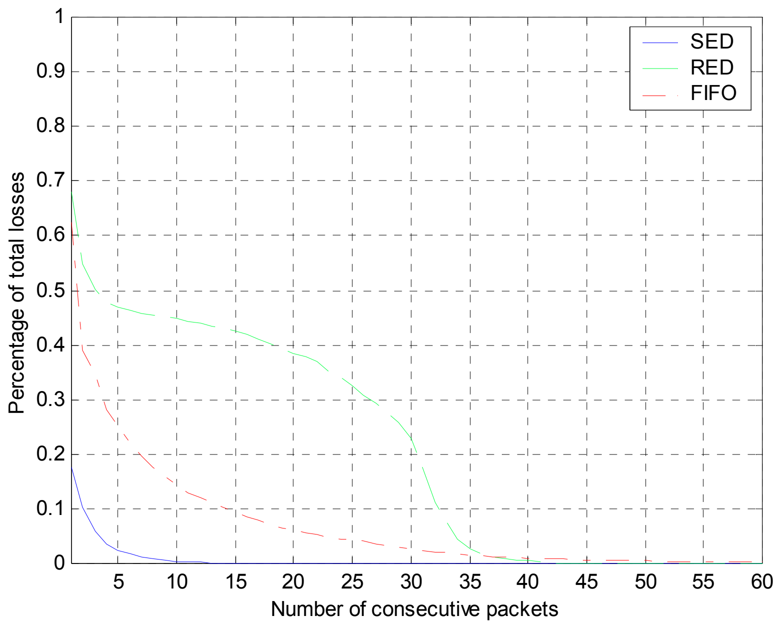


Figure 22. Distribution of consecutive packet losses for SED, RED and FIFO.

D. SED IN A MANET ENVIRONMENT

This section aims to provide results for a real MANET environment with 10 nodes moving within a limited area. Two different area sizes are considered: 10×10 km^2 and 30×30 km^2. For the case of 10×10 km^2, medium traffic load with packet losses of approximately 1% and heavy traffic load with losses of approximately 3% are examined. As FIFO is found to perform better than RED for real-time traffic in all scenarios we tested, hereafter performance comparison is presented between SED and FIFO only.

1. Small Area, Medium Traffic Scenario

Ten nodes and six connections, each one with a source rate of 240 kbps, are simulated. The buffer size is 160 packets, and the SED thresholds are set to (140, 180). Results obtained by averaging two independent simulations for each mobility pattern (pause time), each 13,000-sec long. The average number of packets sent in each simulation run is approximately 770,000 packets per source.

The packet loss, average end-to-end delay and throughput are shown in Figures 23, 24 and 25, respectively. Packet loss for SED is within the range of 0.6% to 1.25% while for FIFO it is 0.65% to 1.54 %.

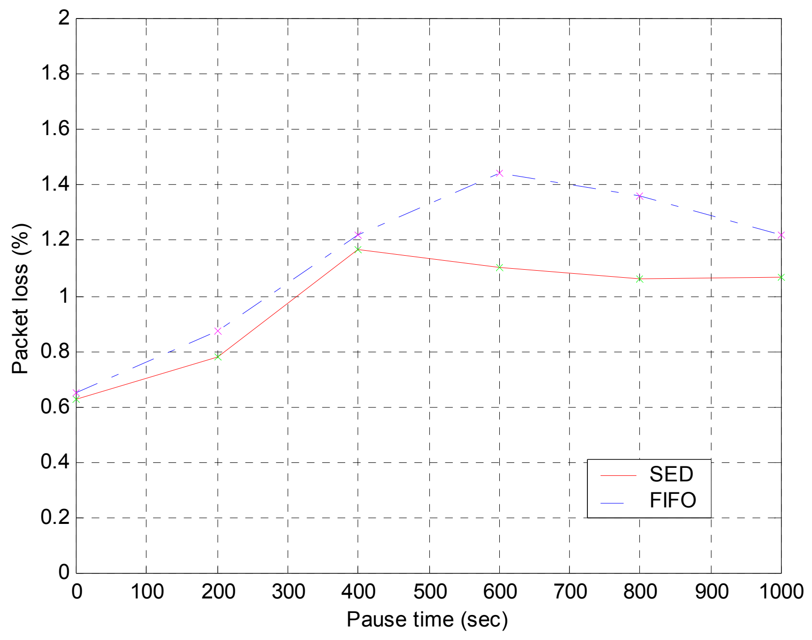


Figure 23. Average packet loss for FIFO and SED (120, 140) for a small area, medium traffic scenario.

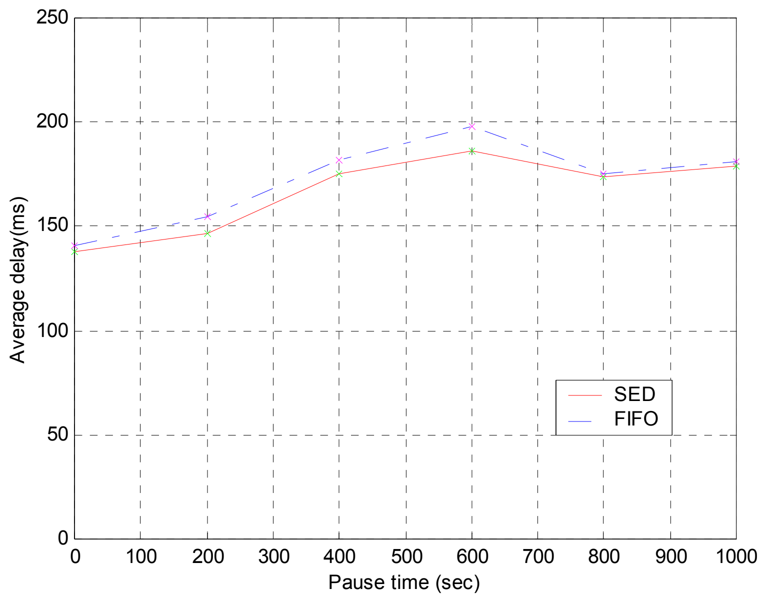


Figure 24. Average end-to-end delay for FIFO and SED (120, 140) for a small area, medium traffic scenario.

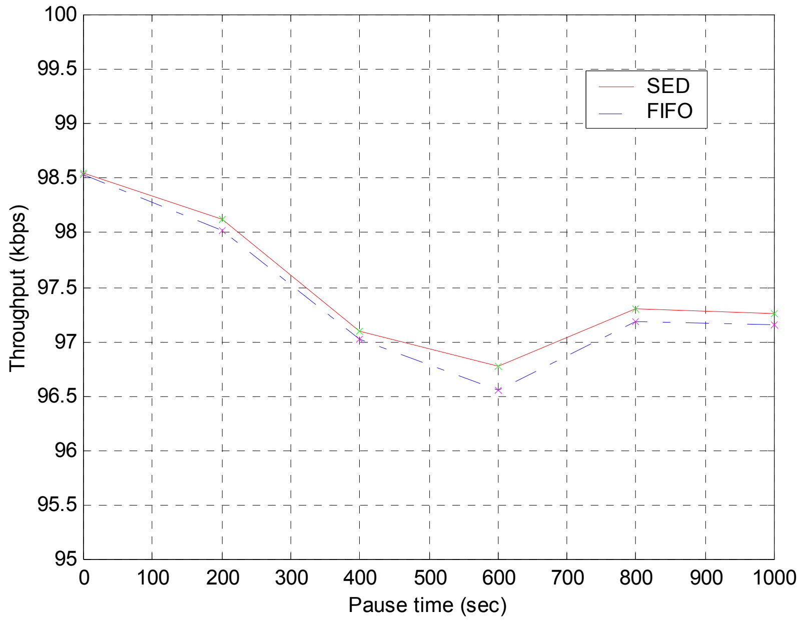


Figure 25. Average throughput for FIFO and SED (120, 140) for a small area, medium traffic scenario.

Packet loss for SED is better than that for FIFO because in FIFO the buffers are more fully occupied; therefore, the effects of exposed and hidden node problems are more severe than in the case of SED. The average end-to-end delay is slightly lower for SED (150 to 200 ms) in all mobility scenarios. Also, SED provides higher throughput than FIFO.

The frequency of packet losses is presented in Figure 26 and the distribution of consecutive packet losses in Figure 27. From these figures, when the packet loss is small, SED almost eliminates both the burst errors and the errors in the intervals of 2-3 packets. In FIFO, burst errors of more than 30 consecutive packets occur while in SED consecutive packet losses are almost eliminated. This large number of consecutive packet losses using FIFO necessitates QoS guarantees in MANETs using a scheme like SED. Considering that the total error in SED is also lower than in FIFO, SED is a better choice for real-time traffic in MANETs.

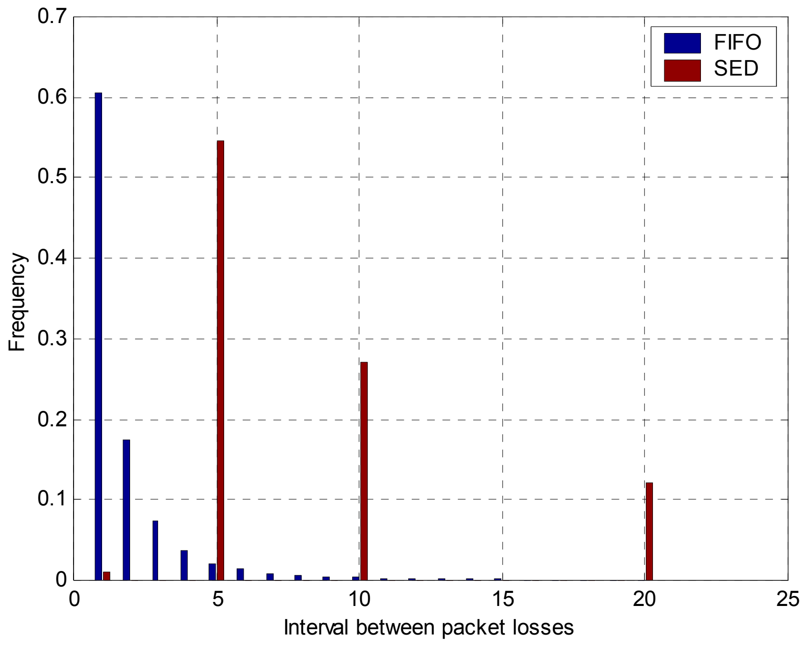


Figure 26. Frequency of packet loss distribution in terms of packet intervals for the first 20 intervals. In each interval the first (left) bar corresponds to FIFO and the second to SED (120, 140) for a small area, medium traffic scenario.

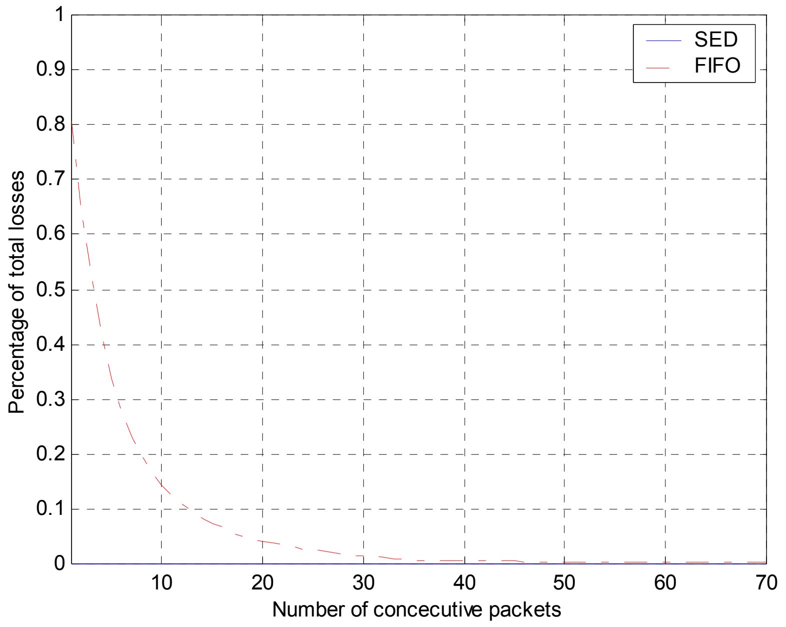


Figure 27. Distribution of consecutive packet losses for FIFO and SED (120, 140) for a small area, medium traffic scenario.

2. Small Area, Heavy Traffic Scenario

Here we consider a geography of 10×10 km^2, 10 nodes and a data rate of 280 kbps per source. The purpose is to investigate the performance of SED in spreading errors under heavy traffic. Plots of the results are obtained by averaging results from three independent simulation runs. Each of the simulation runs is 4,000-sec long, resulting in generation of 280,000 packets per source per simulation run.

Figures 28 and 29 show the packet loss and average delay, respectively. Results generally are similar to those in the previous section. SED provides less error compared to FIFO and has a slightly lower average end-to-end delay. Numerically, both errors and average end-to-end delay are higher in this scenario due to increased congestion in the buffer. Although not shown here, the trends in frequency of packet loss and distribution of consecutive loss plots are similar to those of the small size, medium traffic scenario.

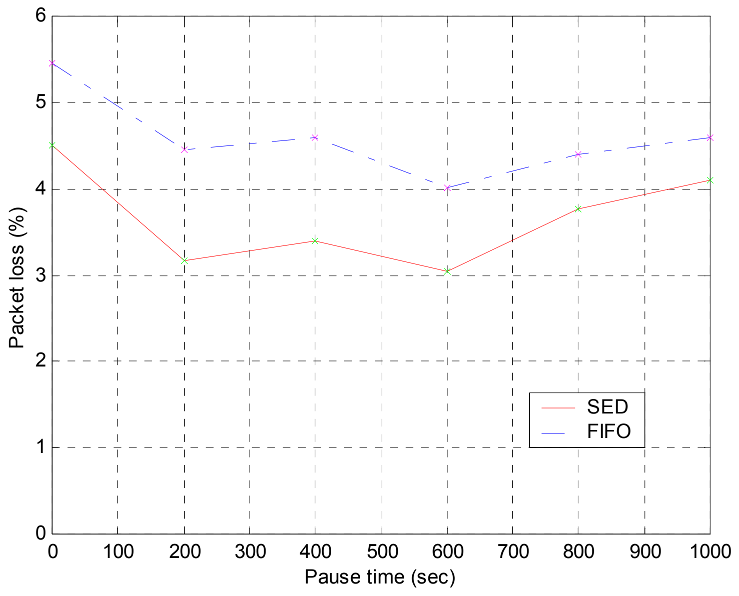


Figure 28. Average packet loss for FIFO and SED (120, 140) for a small area, heavy traffic scenario.

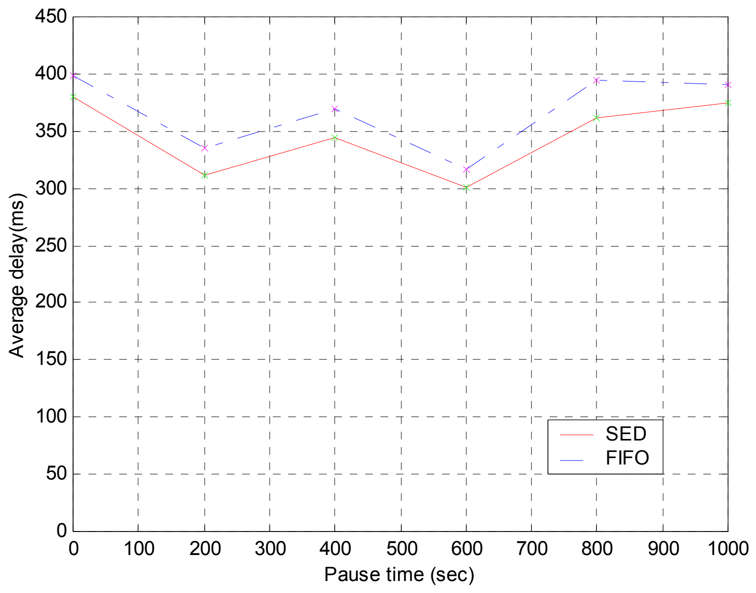


Figure 29. Average end-to-end delay for FIFO and SED (120, 140) for a small area, heavy traffic scenario.

3. Large Area

We now consider 10 nodes in an area of 30×30 km^2 instead of 10×10 km^2 and six active connections with each source transmitting at a rate of 240 kbps. The maximum buffer size is 160 packets, SED's thresholds are (120, 140), and the dropping parameters are 20,10 and 5. Three independent simulations are run for each mobility scenario, each 4,000-sec long; approximately 239,000 packets per source per simulation are generated.

Figures 30 and 31 present the packet loss and average delay. For both metrics, results are worse compared to the small area scenario for SED and FIFO. Packet loss is higher due to the fact that in a large area, nodes can be isolated and unable to maintain routes with other nodes because of their limited maximum transmission range. Average end-to-end delay is larger because the average number of intermediate nodes from the source to the destination is larger. Specifically, packets often travel two or three hops in order to reach the destination. In contrast, in a small area scenario, the destination is mostly one hop away; in the worst-case, two hops away. Furthermore, in this scenario, SED and FIFO provide generally the same average error and end-to-end delay.

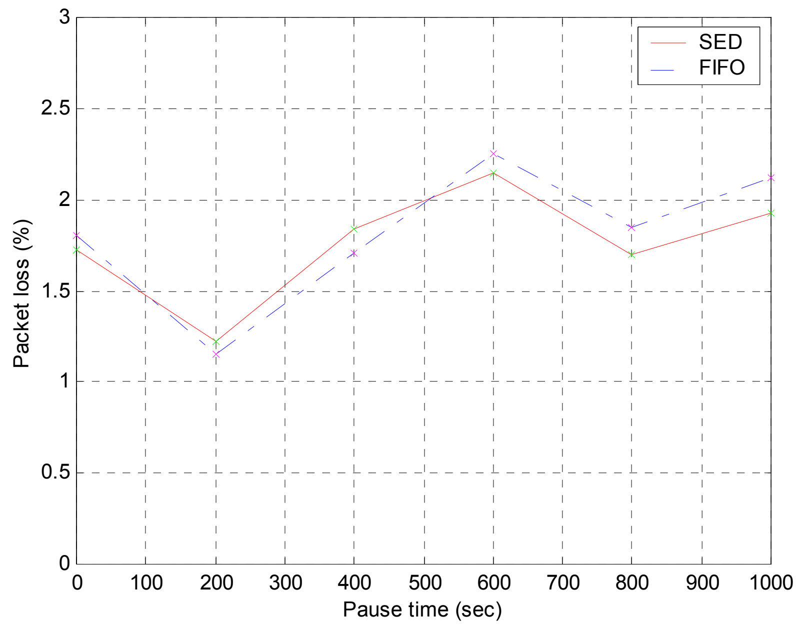


Figure 30. Average packet loss for FIFO and SED (120, 140) for a large area, medium traffic scenario.

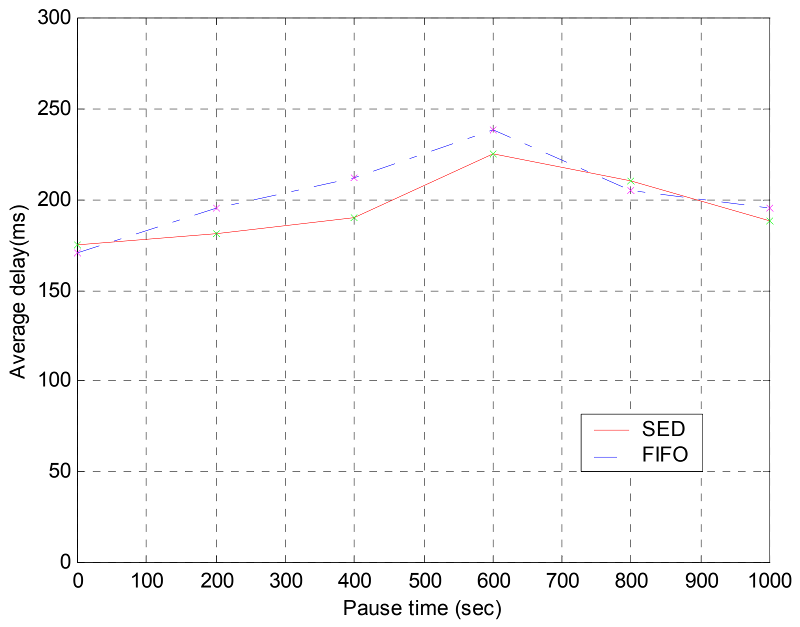


Figure 31. Average end-to-end delay for FIFO and SED (120,140) for a large area, medium traffic scenario.

The frequency of packet losses is shown in Figure 32. The FIFO scheme has 70% of the total packet losses in intervals of at least two consecutive packets while SED has only 8%. In the small area size scenario the corresponding percentages were measured to be 60% and 1% for FIFO and SED, respectively. The majority of the additional packet losses in the large area scenario are due to route losses. For example, when a node does not have any other node within its communication range and has packets to send, the node cannot send these packets and arriving packets may find the node buffer filled, leading to consecutive packet drops.

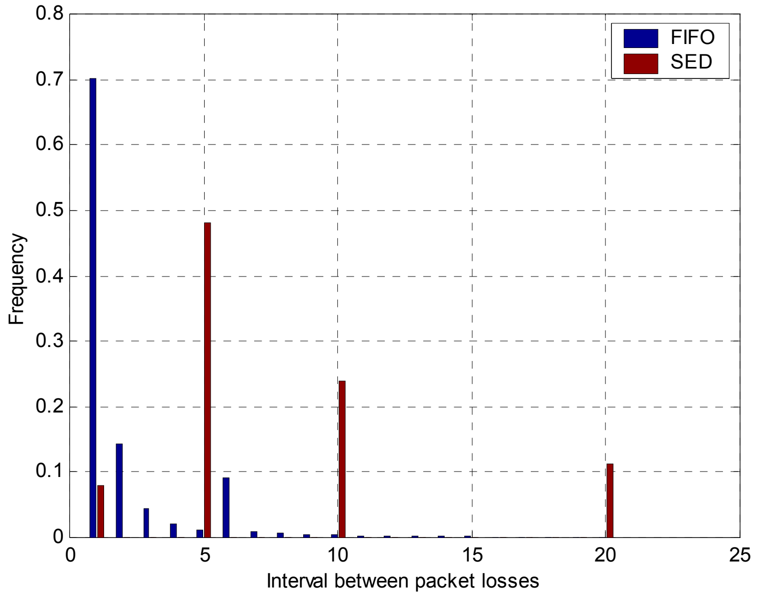


Figure 32. Frequency of packet loss distribution in terms of packets intervals for the first 20 intervals. In each interval the first (left) bar corresponds to FIFO and the second to SED (120, 140) for a large area, medium traffic scenario.

E. SED/IO IN A MANET ENVIRONMENT

This section investigates the proposed SED/IO algorithm in a MANET scenario and compares its performance with that of FIFO. Simulations were run for two different area sizes, 10×10 km^2 and 30×30 km^2. In each simulation scenario, an equal number of high and low priority sources generate an equal amount of traffic. The third threshold for the high priority packets is the maximum buffer capacity.

1. Small Area, Medium Traffic Scenario

The traffic differentiation in SED/IO is achieved by setting the threshold values of the high priority packets (IN packets) to be larger than the corresponding values of the low priority packets (OUT packets). Also, by choosing larger values for selective dropping (SD) parameters for the IN packets, these packets are dropped less frequently. In our simulations the threshold values for the IN packets are (120, 140, 160) and (60, 80, 100) for the OUT packets. The SD parameters are 20, 10 and 5 for the IN packets and 16, 8 and 4 for the OUT packets. The number of connections is six and the rate of each source is 260 kbps. The maximum buffer size is 160 packets. Results represent the average of two independent simulations for each mobility scenario; each run is 12,000-sec long, which results in a total number of packets sent of approximately 775,000 per source.

Figure 33 depicts the packet losses for SED/IO and FIFO. Clearly, SED/IO provides traffic differentiation. The error for the high priority packets is within 0.25-0.6% for the SED/IO algorithm. On the other hand, error rates for FIFO are 2-4 % for both types of traffic. Strictly speaking, the packet loss and delay curves for FIFO high and low priority traffic must not be different. We remark that by averaging these results over a large number of simulations, the curves may indeed converge. Due to computational resource limitation, we were not able to obtain more than three runs in this work. This limitation will have to be addressed in a future effort. However, the SED error rate for the high priority packets is relatively stable for all mobility scenarios. This means that the problems that are attributed to 802.11 do not cause performance degradation in the protected traffic but rather are absorbed by the low priority packets.

The average end-to-end delay is shown in Figure 34. The SED/IO scheme provides significantly lower delay compared to FIFO for both high and low priority traffic. The average delay for the IN packets in SED/IO is 30 to 40 ms higher than the average delay of the OUT packets. This additional delay for IN packets is the result of holding the IN packets longer in the queue than OUT packets. Specifically, OUT packets are dropped earlier and more frequently, which reduces the delay. Compared to FIFO, the average delay also is relatively stable for both types of traffic for SED/IO.

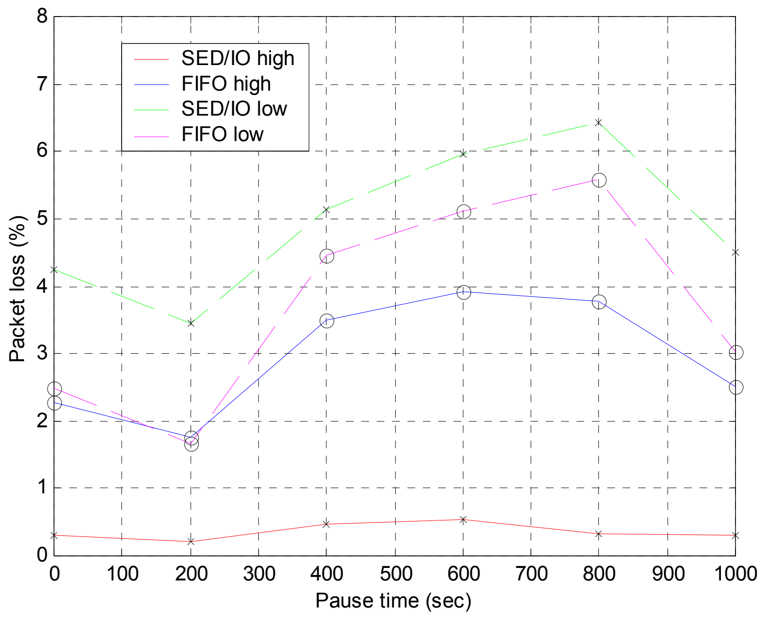


Figure 33. Average packet loss for FIFO and SED/IO for a small area, medium traffic scenario.

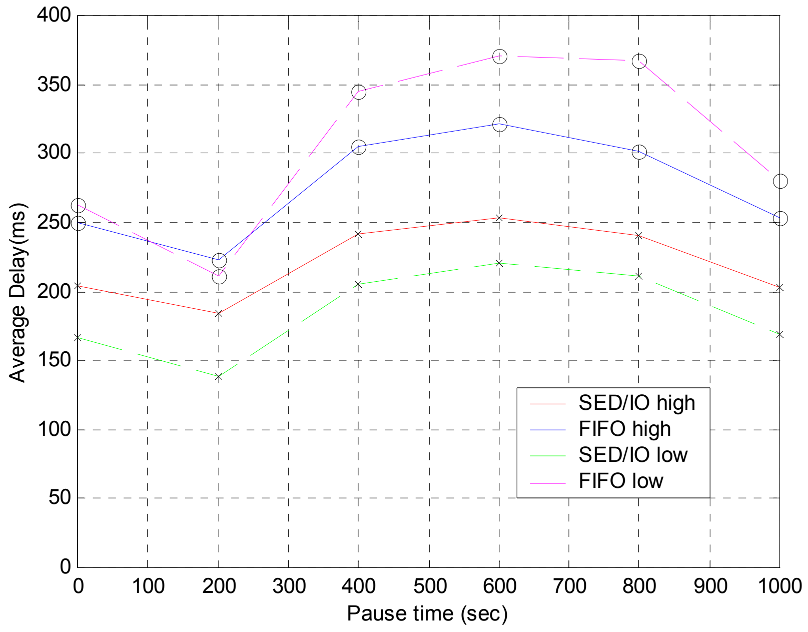


Figure 34. Average end-to-end delay for FIFO and SED/IO for a small area, medium traffic scenario.

Figure 35 provides the frequency of packet losses for both the IN and OUT packets. Results show that SED/IO spreads the error for both high and low priority traffic. For SED/IO low priority traffic, the portion of consecutive packet losses is negligible. Overall, SED/IO provides preferential treatment to high priority traffic (or IN packets).

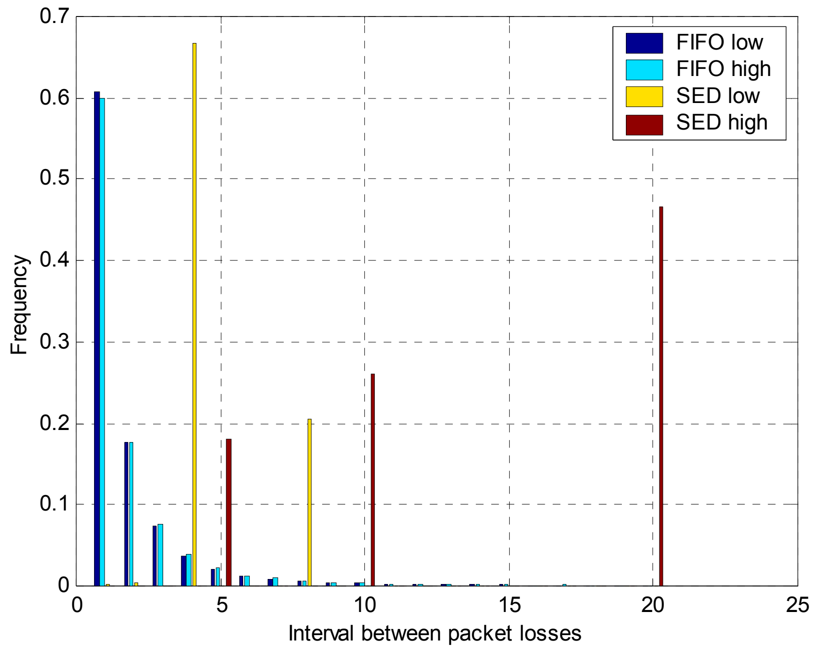


Figure 35. Frequency of packet loss distribution in terms of packets intervals for the first 20 intervals. In each interval the first (left) bar corresponds to FIFO and the second to SED/IO for a small area, medium traffic scenario.

2. Small Area, Heavy Traffic Scenario

Continuing with the scenario in the previous subsection, we now increase the data rate of each source to 300 kbps from 260 kbps. Consequently, we are introducing an additional total traffic of 6×40 = 240 kbps into the network. The thresholds of the OUT packets are changed to (60, 70, 80) from (60, 80, 100) while for the IN packets they remain the same (120, 140, 160). The reason for decreasing the thresholds for OUT packets is to provide protection for the IN packets. Plots are based upon averaging results from three independent simulations for each mobility scenario.

Figures 36 and 37 present the packet loss and average end-to-end delay, respectively. Results of packet loss show that the SED/IO algorithm is able to protect the high priority packets under heavy traffic. Compared to the medium traffic scenario, results indicate that FIFO's performance becomes worse as traffic increases. Results for delay are very similar to those in the medium traffic case except for a net increase in the values.

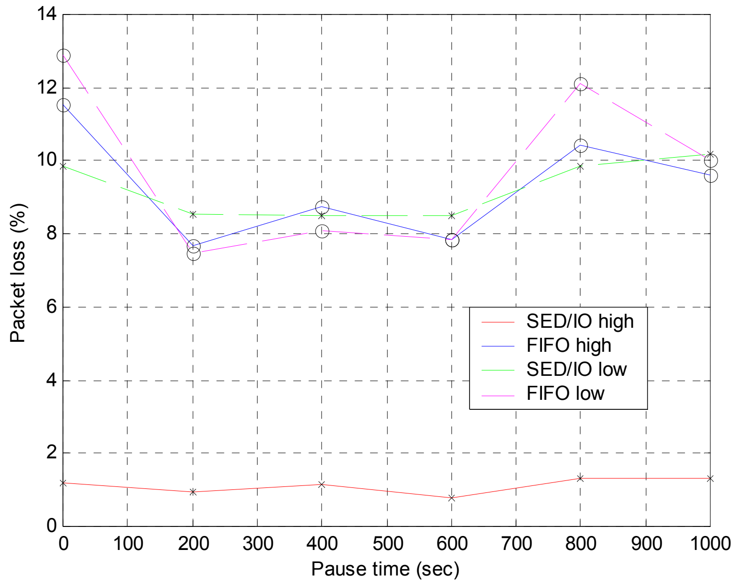


Figure 36. Average packet loss for FIFO and SED/IO for a small area, heavy traffic scenario.

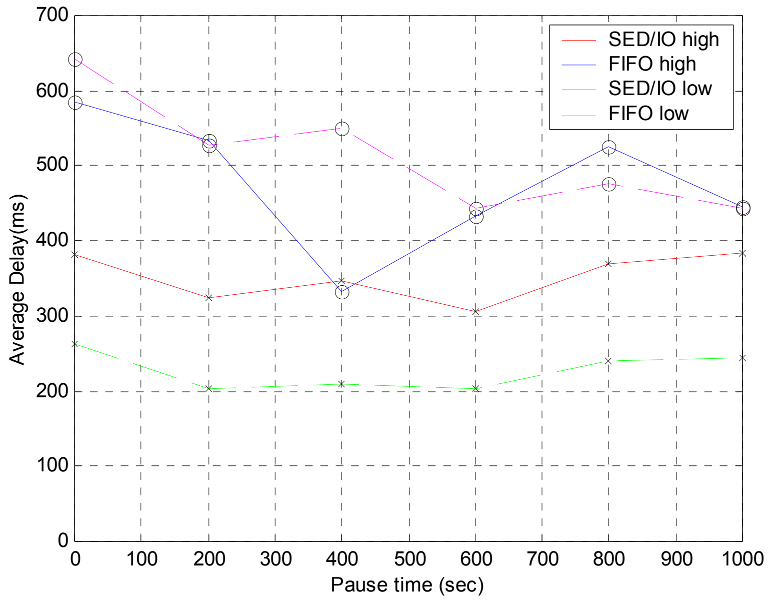


Figure 37. Average end-to-end delay for FIFO and SED/IO for a small area, heavy traffic scenario.

F. SED AND SED/RIO WITH TIMESTAMPS IN A MANET ENVIRONMENT

1. SED with Timestamps

The performance of the SED algorithm with timestamps is compared with that of SED without timestamps and FIFO in this section. SED with timestamps has the advantage of dropping packets that might be too late at the receiver, thus improving its performance over that of SED. However, SED with timestamps requires routers to read the RTP header, which makes this scheme more complex than the simple SED. The objective of the simulations in this section is to demonstrate the improvement in performance achieved by SED with timestamps as a QoS scheme.

An area of 10×10 km^2, ten nodes, and six sources generating traffic at a rate of 260 kbps are used. The maximum buffer capacity is 200 while the thresholds are (140, 180, 200) and the selective parameters are 20, 10 and 5 for both SED algorithms. Three independent simulations were run for each mobility scenarios, each 4,000-sec long.

Figures 38 and 39 depict the packet loss and average end-to-end delay for the three QoS schemes. SED with timestamps provides the lowest error rate and end-to-end delay. Low errors occur because delayed packets are dropped early. The average delay is low because when the queue size exceeds the dropping thresholds, SED with timestamps drops packets that have been in the network longer than others and are likely to be unusable at the receiver.

2. SED/IO with Timestamps

This section provides simulation results for SED/IO with and without timestamps. A geographical area of 10×10 km^2, ten nodes, six sources sending at a rate of 280 kbps and a maximum buffer capacity of 200 packets are used. Threshold values for the IN and OUT packets are (140, 180, 200) and (100, 120, 140), respectively. Results represent averages from three independent simulations for each mobility scenario, each 4,000-sec long.

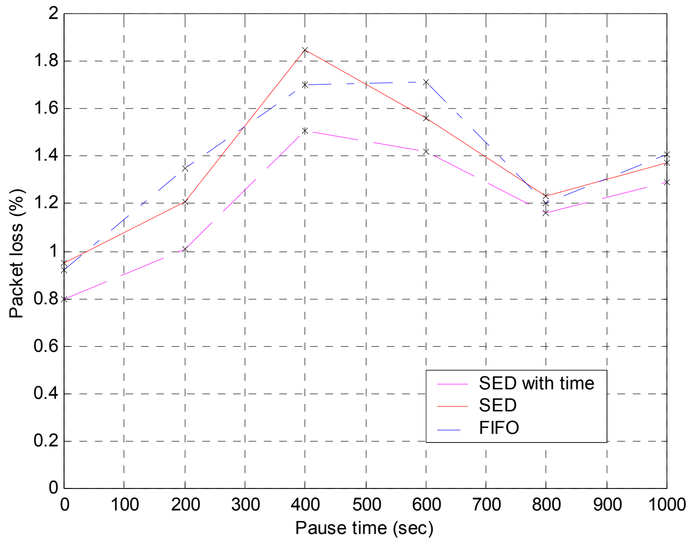


Figure 38. Average packet loss for SED with timestamps, SED without timestamps and FIFO for a small area.

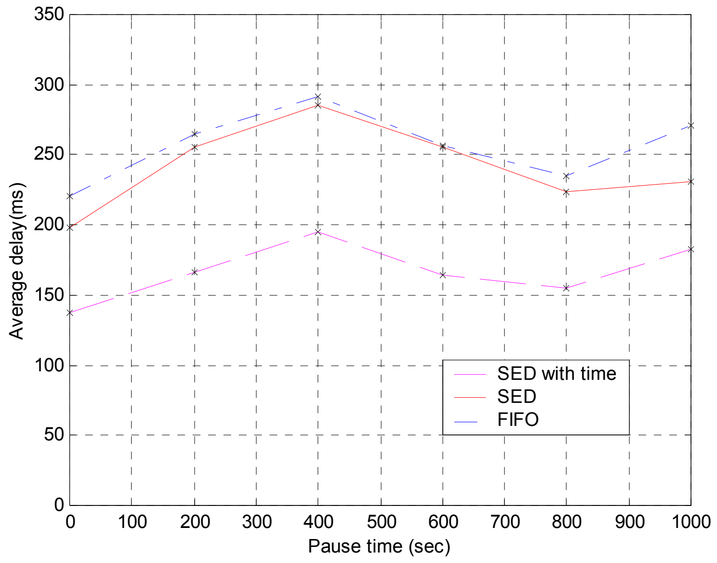


Figure 39. Average end-to-end delay for SED with timestamps, SED without timestamps and FIFO for a small area.

Figures 40 and 41 show the packet loss and average end-to-end delay for each type of traffic for all schemes. SED/IO with timestamps performs better than SED without timestamps in both types of traffic. The IN packets have approximately 0.02% less error while the average delay is decreased by more than 60 ms compared to SED/IO without timestamps. For the OUT packets, the improvement in packet losses in SED/IO using timestamps is smaller and the improvement in average delay is approximately 35 ms. Consequently, SED/IO using timestamps provides significant performance benefits and these advantages are greater for the high priority packets.

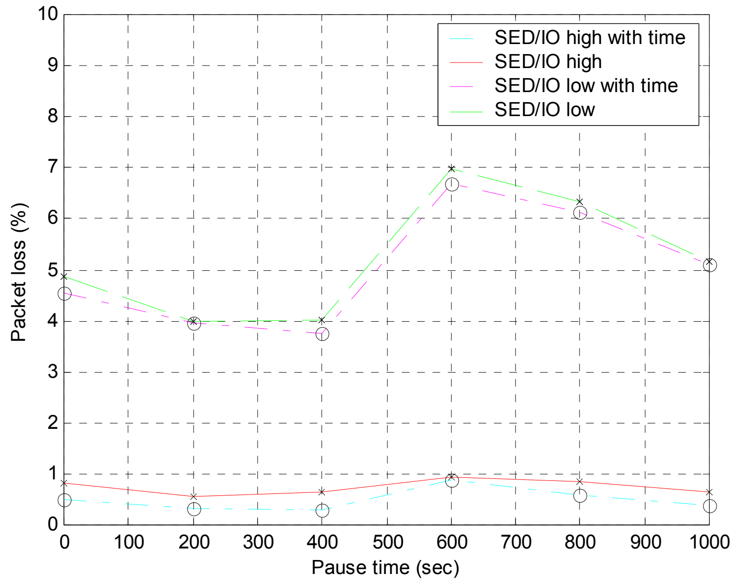


Figure 40. Average packet loss for SED/IO with and without timestamps.

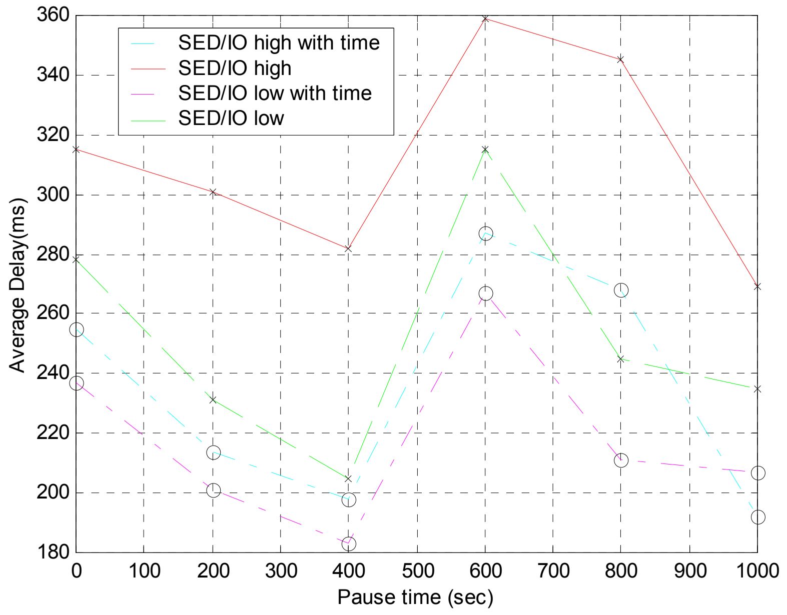


Figure 41. Average end-to-end delay for SED/IO with and without timestamps.

F. BIT DROPPING SCHEME IN A MANET ENVIRONMENT

Results on bit-dropping have shown that significant bandwidth savings are possible by dropping bits from each CELP encoded voice packet. In particular, by losing 24 bits of each FS1016 voice frame, voice quality degradation is tolerable. As a result, during periods of congestion in the buffer, bits can be dropped to reduce congestion.

The purpose of the simulations in this section is to show the benefits of bit-dropping schemes. Simulations were run for both SED and the proposed bit-dropping scheme. Note that the bit-dropping scheme is not actually implemented but rather is approximated. Specifically, packets are dropped instead of bits but these packets correspond to the total number of bits dropped if the proposed scheme were implemented.

The simulation parameters are the same as in the SED heavy traffic scenario. It includes a small area of 10×10 km^2, ten nodes and six sources that generate data at 280 kbps. The maximum buffer capacity is 160 packets and a threshold $\Theta = 120$ is set for dropping the bits. It is assumed that when a packet arrives and the queue size exceeds this threshold, 24 bits are dropped from six packets. Thus, the total number of bits dropped when an arriving packet finds the queue size to be above the threshold is $24 \times 6 = 144$ bits, which correspond to a whole FS1016 frame, and it allows the approximation of dropping these bits by discarding a packet. Although this simulation configuration does not exactly represent the proposed bit-dropping scheme, the objective of demonstrating the benefits of bit-dropping schemes is accomplished. Three 4,000-sec long independent simulations were run to obtain results. The average number of packets sent by each source per simulation run is 259,000, and the number of mobility scenarios is six as in the previous sections.

Packet loss for SED is 3 to 4% as in the simulations for the SED heavy traffic scenario of Chapter VI Section D.2 (Figure 28). In contrast, in all simulations, packet loss for the bit-dropping scheme is below 10^{-3} or, in other words, one packet out of 1,000 is lost at a maximum. The reader is cautioned that these are preliminary results and more rigorous simulations are required to draw final conclusions.

Figure 42 depicts the average end-to-end delay for SED and bit dropping QoS schemes. The bit-dropping scheme provides significantly lower delay (approximately 50 ms in average) compared to SED. This is because dropping bits smoothes the burstiness of traffic, thus speeding up the packet service time during periods of congestion in the queue.

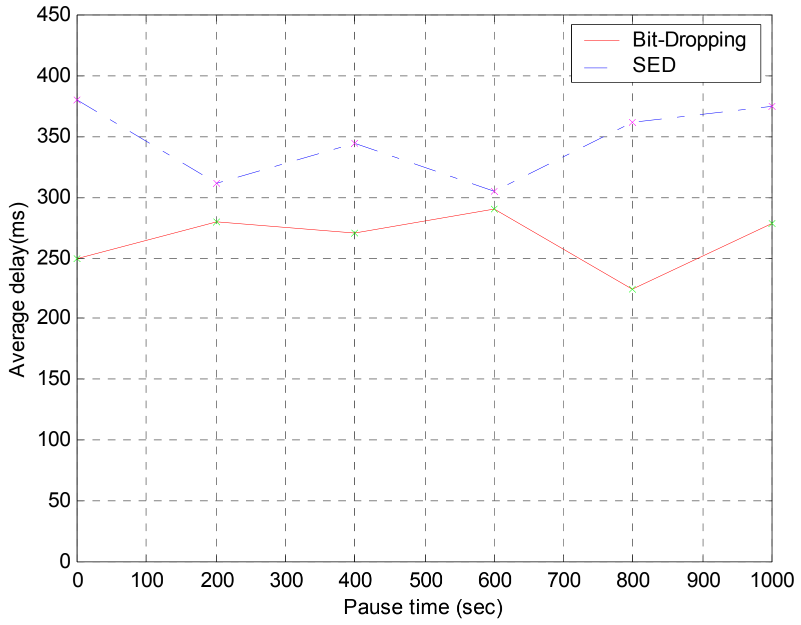


Figure 42. Average end-to-end delay for bit-dropping and SED QoS schemes.

H. SUMMARY

Several simulations were run for the evaluation of SED, the proposed packet dropping algorithm. SED's performance in terms of packet loss, end-to-end delay, and distribution of packet losses was compared with that of FIFO and RED QoS schemes. Additionally, SED's extensions, SED/IO and SED with timestamps, were evaluated using extensive simulations. Results indicate that both SED and its extensions provide significant performance benefits for real-time traffic. Finally, preliminary results for bit-dropping indicate that performance can be further improved by spreading the error at the bit level.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

A. DISCUSSION OF RESULTS

In this thesis, new active queue management schemes using packet and bit dropping techniques were investigated. The main objective was the development of new QoS schemes that improve the performance and provide service differentiation for real-time traffic in MANETs.

The proposed packet dropping technique, called the Selective Early Discard (SED) algorithm, was found to be effective in improving the packet loss and delay performance for real-time traffic. The SED algorithm selectively drops packets according to a specific pattern using three selective dropping parameters and three thresholds in order to spread the error as much as possible. The minimum threshold is used mainly for the control of the queuing delay by dropping packets early. On the other hand, the maximum threshold is used to drop packets when necessary just before the buffer overflow occurs in order to spread the error. A parameter sensitivity study was performed to experimentally determine the thresholds and examine the trade offs involved. Using extensive simulations, the performance of the SED algorithm was studied and compared with that of the Random Early Discard (RED) and First In First Out (FIFO) queuing schemes. The performance metrics of interest were packet loss, end-to-end delay, and frequency and distribution of packet losses.

The SED algorithm, compared to the RED and FIFO schemes, provides better performance by reducing burst errors. Specifically, it was found that SED minimizes the burst errors due to buffer overflow, adds only a small amount to the total error by dropping packets early, and decreases the average delay. In the simulation experiments with voice packets using the FS1016 CELP coder, results indicated a significant degradation in voice quality if more than 1 packet is lost in a sequence of five packets. Simulation results showed that the SED algorithm minimizes, as much as possible, the occurrence of 2 or more packet losses in a sequence of five packets. Furthermore, in the MANET simulation scenarios, SED keeps all the above-mentioned benefits while also giving lower average error compared to FIFO. A possible reason is that by dropping packets early, the SED algorithm smoothes the hidden and exposed node problems of the 802.11 MAC protocol.

The SED algorithm was extended in order to provide service differentiation. This scheme is called SED/IO (SED with IN and OUT traffic classes) and utilizes two independent SED algorithms, one for each traffic class. Results indicate that SED/IO is able to provide service differentiation by using different selective dropping parameters for the two classes of packets.

Another variation of SED utilizes the timestamp field in the RTP header. The purpose is for the intermediate node to drop packets that might be unusable at the destination due to excessive delay. Simulations show that this scheme not only decreases the average end-to-end delay but also provides lower error compared to SED and FIFO.

Finally, a bit-dropping technique was proposed as another possible QoS scheme that aims to spread errors at the bit level (rather than spreading the error at the packet level as in SED). This scheme is more complex than SED and requires access to payload of the packet in the intermediate nodes. Simulations showed that by dropping 24 bits of each FS1016 frame, the degradation in voice is acceptable. Preliminary simulations indicated that significant bandwidth and performance advantages can be gained as a result of dropping bits.

B. SUGGESTIONS FOR FUTURE WORK

Although the simulation results in this thesis are based on voice traffic, the SED algorithm can be extended to other types of real-time traffic in which a certain amount of tolerance in terms of packet losses is permissible. In this connection, it will be an interesting effort to investigate the improvement in performance possible for real-time video traffic and determine general guidelines in dropping packets according to specific characteristics.

In this thesis, the main effort was put into the development of efficient active queue management QoS schemes for real-time traffic only. Nevertheless, in both MANETs and wired networks, a large amount of non-real-time traffic, such as e-mail and ftp, exists. A future study may consider the mixed traffic scenario and apply SED or its variants to the real-time traffic and an equivalent algorithm to manage the non-real-time traffic.

The bit-dropping algorithm presented in the thesis needs to be further studied. In its current form, the algorithm requires access to payload in the packet, which makes the implementation in an intermediate node quite complex. Ways to simplify it by embedding the packets at the source with predefined congestion states are to be explored. The guidelines for selective drop parameters and thresholds need to be developed.

In this thesis, the packet and bit dropping studies examined only buffer losses with respect to voice quality. However, the wireless environment introduces additional impairments, such as bit errors that lead to route losses and header corruption. These issues along with a more in-depth study of MAC layer constraints are of considerable interest.

The SED algorithm could be implemented in future networks in association with the error concealment schemes for real-time traffic [35]. In these techniques, the missing packets are recovered by processing previous packets; as a result, having an algorithm like SED to eliminate burst errors is a necessity.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A – SED AND SED/RIO C++ CODE FILES

The following C++ code is used in creating MANET simulations for SED and its variants and RED algorithm in NS2. Main modifications and additions by the author are highlighted in bold font.

```
/* -*-  Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t -*- */
/*
 * Copyright (c) 1990-1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by the Computer Systems
 *  Engineering Group at Lawrence Berkeley Laboratory.
 * 4. Neither the name of the University nor of the Laboratory may be used
 *    to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *
 * Here is one set of parameters from one of my simulations:
 *
 * ed [ q_weight=0.002 thresh=5 linterm=30 maxthresh=15
 *         mean_pktsize=500 dropmech=random-drop queue-size=60
 *         plot-file=none bytes=false doubleq=false dqthresh=50
 *     wait=true ]
 *
 * 1/"linterm" is the max probability of dropping a packet.
 * There are different options that make the code
 * more messy that it would otherwise be.  For example,
 * "doubleq" and "dqthresh" are for a queue that gives priority to
 *   small (control) packets,
 * "bytes" indicates whether the queue should be measured in bytes
 *   or in packets,
 * "dropmech" indicates whether the drop function should be random-drop
 *   or drop-tail when/if the queue overflows, and
 *   the commented-out Holt-Winters method for computing the average queue
 *   size can be ignored.
 * "wait" indicates whether the gateway should wait between dropping
 *   packets.
 *
 * @(#) $Header: /usr/src/mash/repository/vint/ns-2/rio.h,v 1.14 1998/06/27 01:24:29 gnguyen
Exp $ (LBL)
 */

#ifndef hannan_debug
#define hannan_debug
#endif

#ifndef ns_rio_h
#define ns_rio_h

//#define ACKOUT  /* this is used to switch whether mark ACK as IN or OUT */
#define ACKOUT

#ifndef INOUT
#define INOUT

/* ACK are markded as IN */
#ifdef ACKIN
#define IN 0
#define OUT 1
#endif

/* ACK are markded as OUT */
#ifdef ACKOUT
#define IN  1
#define OUT 0
#endif
#endif

#include "../queue.h"
#include "../red.h"

class LinkDelay;

class RIOQueue : public Queue {
 public:
   RIOQueue();
   void enque(Packet* pkt);
   Packet* deque();
 protected:
   int command(int argc, const char*const* argv);
   int decide_droptype(Packet* pkt, edp* edp_, edv* edv_);  /* hannan, decide the drop type of the
packet */
   virtual Packet *pickPacketForECN(Packet* pkt);
   virtual Packet *pickPacketToDrop();
```

```
    void reset();
    void run_estimator(int nqueued, int m, edp* edp_, edv* edv_);
    int drop_early(Packet* pkt, edp* edp_, edv* edv_);

    LinkDelay* link_;   /* outgoing link */
    int fifo_;      /* fifo queue? */
    PacketQueue *q_;    /* underlying (usually) FIFO queue */

    int bcount_;    /* byte count */
    int qib_;   /* bool: queue measured in bytes? */
    NsObject* de_drop_; /* drop_early target */

    Tcl_Channel tchan_; /* place to write trace records */
    TracedInt curq_;    /* current qlen seen by arrivals */
    void trace(TracedVar*); /* routine to write trace records */

    /*
     * Static state.
     */
    int drop_tail_;     /* drop-tail */
    int drop_front_;    /* drop-from-front */
    int drop_rand_;     /* drop-tail, or drop random? */

    edp INedp_;   /* hannan RED params of IN packets */
    edp OUTedp_;  /* hannan RED params of OUT packets */

    /*
     * Dynamic state.
     */
    int idle_;      /* queue is idle? */
    double idletime_;   /* if so, since this time */
    edv INedv_;       /* hannan early-drop variables of IN packets*/
    edv OUTedv_;     /* hannan early-drop variables of OUT packets*/
    int INnum_;     /* hannan # of IN packets in the queue */
    TracedInt tag_;       /* hannan to trace the tag of packet IN/OUT */

    void print_edp(edp* edp_);   // for debugging
    void print_edv(edv* edv_);   // for debugging
};

#endif
```

```
/* -*-  Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t -*- */
/*
 * Copyright (c) 1990-1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *  This product includes software developed by the Computer Systems
 *  Engineering Group at Lawrence Berkeley Laboratory.
 * 4. Neither the name of the University nor of the Laboratory may be used
 *    to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *
 * Here is one set of parameters from one of Sally's simulations
 * (this is from tcpsim, the older simulator):
 *
 * ed [ q_weight=0.002 thresh=5 linterm=30 maxthresh=15
 *         mean_pktsize=500 dropmech=random-drop queue-size=60
 *         plot-file=none bytes=false doubleq=false dqthresh=50
 *     wait=true ]
 *
 * 1/"linterm" is the max probability of dropping a packet.
 * There are different options that make the code
 * more messy that it would otherwise be.  For example,
 * "doubleq" and "dqthresh" are for a queue that gives priority to
 *   small (control) packets,
 * "bytes" indicates whether the queue should be measured in bytes
 *   or in packets,
 * "dropmech" indicates whether the drop function should be random-drop
 *   or drop-tail when/if the queue overflows, and
 *   the commented-out Holt-Winters method for computing the average queue
 *   size can be ignored.
 * "wait" indicates whether the gateway should wait between dropping
 *   packets.
 */
```

```
#ifndef lint
static const char rcsid[] =
   "@(#) $Header: /usr/src/mash/repository/vint/ns-2/fqmm/rio.cc,v 1.34 1999/01/07 19:01:57
sfloyd Exp $ (LBL)";
#endif

#define hannan_debug

#include <math.h>
#include <string.h>
#include <sys/types.h>
#include "../template.h"
#include "../random.h"
#include "../flags.h"
#include "../delay.h"
#include "../ip.h"
#include "rio.h"
```
#include "../rtp.h" //in order RTP header to be read
#include <iostream.h>

```
static class RIOClass : public TclClass {
public:
   RIOClass() : TclClass("Queue/RIO") {}
   TclObject* create(int, const char*const*) {
      return (new RIOQueue);
   }
} class_rio;

RIOQueue::RIOQueue() : link_(NULL), bcount_(0), de_drop_(NULL),
   tchan_(0), idle_(1)
{
   bind_bool("bytes_", &INedp_.bytes);       // boolean: use bytes?
   OUTedp_.bytes = INedp_.bytes;
   bind_bool("queue-in-bytes_", &qib_);        // boolean: q in bytes?

   // set parameters for IN packets
   bind("in_minthresh_", &INedp_.th_min);          // minthresh
   bind("in_maxthresh_", &INedp_.th_max);       // maxthresh
   bind("in_q_weight_", &INedp_.q_w);           // for EWMA
   bind("in_linterm_", &INedp_.max_p_inv);      // inverse of the max drop prob.

   // set parameters for OUT packets
   bind("out_minthresh_", &OUTedp_.th_min);          // minthresh
   bind("out_maxthresh_", &OUTedp_.th_max);       // maxthresh
   bind("out_q_weight_", &OUTedp_.q_w);           // for EWMA
   bind("out_linterm_", &OUTedp_.max_p_inv);      // inverse of the max drop prob.

   bind("mean_pktsize_", &INedp_.mean_pktsize);  // avg pkt size
   OUTedp_.mean_pktsize = INedp_.mean_pktsize;
   bind_bool("wait_", &INedp_.wait);
   OUTedp_.wait = INedp_.wait;
   bind_bool("setbit_", &INedp_.setbit);     // mark instead of drop
   OUTedp_.setbit = INedp_.setbit;
   bind_bool("drop-tail_", &drop_tail_);       // drop last pkt
   bind_bool("drop-front_", &drop_front_);     // drop first pkt
   bind_bool("drop-rand_", &drop_rand_);       // drop pkt at random

   bind("in_ave_", &INedv_.v_ave); // hannan average queue size of IN pkt
   bind("out_ave_", &OUTedv_.v_ave); // hannan average queue size of OUT pkt
   bind("in_prob1_", &INedv_.v_prob1);  //hannan dropping prob. of IN pkt
   bind("out_prob1_", &OUTedv_.v_prob1);  //hannan dropping prob. of OUT pkt

   bind("curq_", &curq_);    // current queue size
   bind("tag_", &tag_);      // tag of the packet
   q_ = new PacketQueue();             // underlying queue
   pq_ = q_;
   reset();

#ifdef notdef
print_edp(&INedp_);
print_edv(&INedv_);
print_edp(&OUTedp_);
print_edv(&OUTedv_);
#endif

}

void RIOQueue::reset()
{
   /*
    * If queue is measured in bytes, scale min/max thresh
    * by the size of an average packet (which is specified by user).
    */

   if (qib_) {
      INedp_.th_min *= INedp_.mean_pktsize;
      INedp_.th_max *= INedp_.mean_pktsize;
      OUTedp_.th_min *= OUTedp_.mean_pktsize;
      OUTedp_.th_max *= OUTedp_.mean_pktsize;
   }

   /*
    * Compute the "packet time constant" if we know the
    * link bandwidth.  The ptc is the max number of (avg sized)
    * pkts per second which can be placed on the link.
    * The link bw is given in bits/sec, so scale mean psize
    * accordingly.
    */

   if (link_) {
      INedp_.ptc = link_->bandwidth() /
         (8. * INedp_.mean_pktsize);
      OUTedp_.ptc = INedp_.ptc;
   }
   INedv_.v_ave = 0.0;
   INedv_.v_slope = 0.0;
   INedv_.count = 0;
   INedv_.count_bytes = 0;
   INedv_.old = 0;
```

```
    INedv_.v_a = 1 / (INedp_.th_max - INedp_.th_min);
    INedv_.v_b = - INedp_.th_min / (INedp_.th_max - INedp_.th_min);

    OUTedv_.v_ave = 0.0;
    OUTedv_.v_slope = 0.0;
    OUTedv_.count = 0;
    OUTedv_.count_bytes = 0;
    OUTedv_.old = 0;
    OUTedv_.v_a = 1 / (OUTedp_.th_max - OUTedp_.th_min);
    OUTedv_.v_b = - OUTedp_.th_min / (OUTedp_.th_max - OUTedp_.th_min);

    idle_ = 1;
    if (&Scheduler::instance() != NULL)
        idletime_ = Scheduler::instance().clock();
    else
        idletime_ = 0.0; /* sched not instantiated yet */
    Queue::reset();

    bcount_ = 0;
    INnum_ = 0;
}

/*
 * Compute the average queue size.
 * The code contains two alternate methods for this, the plain EWMA
 * and the Holt-Winters method.
 * nqueued can be bytes or packets
 * We add two incoming parameters to the method
 */
void RIOQueue::run_estimator(int nqueued, int m, edp* edp_, edv* edv_)
{
    float f, f_sl, f_old;

    f = edv_->v_ave;
    f_sl = edv_->v_slope;
#define RED_EWMA
#ifdef RED_EWMA
    while (--m >= 1) {
        f_old = f;
        f *= 1.0 - edp_->q_w;
    }
    f_old = f;
    f *= 1.0 - edp_->q_w;
    f += edp_->q_w * nqueued;
#endif
#ifdef RED_HOLT_WINTERS
    while (--m >= 1) {
        f_old = f;
        f += f_sl;
        f *= 1.0 - edp_->q_w;
        f_sl *= 1.0 - 0.5 * edp_->q_w;
        f_sl += 0.5 * edp_->q_w * (f - f_old);
    }
    f_old = f;
    f += f_sl;
    f *= 1.0 - edp_->q_w;
    f += edp_->q_w * nqueued;
    f_sl *= 1.0 - 0.5 * edp_->q_w;
    f_sl += 0.5 * edp_->q_w * (f - f_old);
#endif
    edv_->v_ave = f;
    edv_->v_slope = f_sl;
}

/*
 * Return the next packet in the queue for transmission.
 */
Packet* RIOQueue::deque()
{
    Packet *p;
    p = q_->deque();
    if (p != 0) {
        idle_ = 0;
        bcount_ -= ((hdr_cmn*)p->access(off_cmn_))->size();
        /*
         * if is IN packet, decrease the INnum_
         */
         hdr_ip* iph=hdr_ip::access(p);
         if (iph->prio_ == IN) INnum_--;
// quite strange, that sometimes, deque was called before enque, and packetqueue is not empty
// I just let INnum = 0, if it is negative
                if (INnum_ <0) INnum_ =0;
    } else {
        idle_ = 1;
        // deque() may invoked by Queue::reset at init
        // time (before the scheduler is instantiated).
        // deal with this case
        if (&Scheduler::instance() != NULL)
            idletime_ = Scheduler::instance().clock();
        else
            idletime_ = 0.0;
    }
    return (p);
}

/*
 * should the packet be dropped/marked due to a probabilistic drop?
 */

int
RIOQueue::drop_early(Packet* pkt, edp* edp_, edv* edv_)
{
    hdr_cmn* ch = (hdr_cmn*)pkt->access(off_cmn_);

    double p = edv_->v_a * edv_->v_ave + edv_->v_b;
    p /= edp_->max_p_inv;
    edv_->v_prob1 = p;
    if (edv_->v_prob1 > 1.0)
        edv_->v_prob1 = 1.0;
    double count1 = edv_->count;
    if (edp_->bytes)
        count1 = (double) (edv_->count_bytes/edp_->mean_pktsize);
    if (edp_->wait) {
        if (count1 * p < 1.0)
            p = 0.0;
        else if (count1 * p < 2.0)
            p /= (2 - count1 * p);
        else
            p = 1.0;
    } else {
        if (count1 * p < 1.0)
            p /= (1.0 - count1 * p);
        else
            p = 1.0;
    }
    if (edp_->bytes && p < 1.0) {
        p = p * ch->size() / edp_->mean_pktsize;
    }
    if (p > 1.0)
        p = 1.0;
    edv_->v_prob = p;

    // drop probability is computed, pick random number and act
    double u = Random::uniform();
    if (u <= edv_->v_prob) {
        // DROP or MARK
        edv_->count = 0;
        edv_->count_bytes = 0;
        hdr_flags* hf = (hdr_flags*)pickPacketForECN(pkt)->access(off_flags_);
        if (edp_->setbit && hf->ect()) {
            hf->ce() = 1;   // mark Congestion Experienced bit
            return (1);     //
        } else {
            return (1); // drop
        }
    }
    return (0);         // no DROP/mark
}

/*
 * Pick packet for early congestion notification (ECN). This packet is then
 * marked or dropped. Having a separate function do this is convenient for
 * supporting derived classes that use the standard RIO algorithm to compute
 * average queue size but use a different algorithm for choosing the packet for
 * ECN notification.
 */
Packet*
RIOQueue::pickPacketForECN(Packet* pkt)
{
    return pkt; /* pick the packet that just arrived */
}

/*
 * Pick packet to drop. Having a separate function do this is convenient for
 * supporting derived classes that use the standard RIO algorithm to compute
 * average queue size but use a different algorithm for choosing the victim.
 */
```

```
/* we should assure that the packet picked is the of the same class as that
of arrival, IN packet will induce dropping of IN packets, and OUT packet
will induce dropping of OUT packets */
/* we use the default the incoming packet itself, so assure this point */

Packet*
RIOQueue::pickPacketToDrop()
{
int vic;
   int victim;

   if (drop_front_)
      victim = min(1, q_->length()-1);
   else if (drop_rand_)
      victim = Random::integer(q_->length());
   else           // default is drop_tail_
      victim = q_->length() - 1;

   return(q_->lookup(victim));
}


/*
 * Receive a new packet arriving at the queue.
 * The average queue size is computed.  If the average size
 * exceeds the threshold, then the dropping probability is computed,
 * and the newly-arriving packet is dropped with that probability.
 * The packet is also dropped if the maximum queue size is exceeded.
 *
 * "Forced" drops mean a packet arrived when the underlying queue was
 * full or when the average q size exceeded maxthresh.
 * "Unforced" means a RIO random drop.
 *
 * For forced drops, either the arriving packet is dropped or one in the
 * queue is dropped, depending on the setting of drop_tail_.
 * For unforced drops, the arriving packet is always the victim.
 */

#define DTYPE_NONE  0   /* ok, no drop */
#define DTYPE_FORCED    1   /* a "forced" drop */
#define DTYPE_UNFORCED  2   /* an "unforced" (random) drop */

int RIOQueue::decide_droptype(Packet* pkt, edp* edp_, edv* edv_)
{
   /*
    * DROP LOGIC:
    *  q = current q size, ~q = averaged q size
    *  1> if ~q > maxthresh, this is a FORCED drop
    *  2> if minthresh < ~q < maxthresh, this may be an UNFORCED drop
    *  3> if (q+1) > hard q limit, this is a FORCED drop
    */
   hdr_cmn* ch = (hdr_cmn*)pkt->access(off_cmn_);

   register double qavg = edv_->v_ave;
   int droptype = DTYPE_NONE;
   int qlen = qib_ ? bcount_ : q_->length();
```

```
    int qlim = qib_ ? (qlim_ * edp_->mean_pktsize) : qlim_;

    curq_ = qlen;   // helps to trace queue during arrival, if enabled

    if (qavg >= edp_->th_min && qlen > 1) {
        if (qavg >= edp_->th_max) {
            droptype = DTYPE_FORCED;
        } else if (edv_->old == 0) {
            /*
             * The average queue size has just crossed the
             * threshold from below to above "minthresh", or
             * from above "minthresh" with an empty queue to
             * above "minthresh" with a nonempty queue.
             */
            edv_->count = 1;
            edv_->count_bytes = ch->size();
            edv_->old = 1;
        } else {
            droptype = DTYPE_UNFORCED;
        }
    } else {
        /* No packets are being dropped.  */
        edv_->v_prob = 0.0;
        edv_->old = 0;
    }
    if (qlen >= qlim) {
        // see if we've exceeded the queue size
        droptype = DTYPE_FORCED;
    }
    return droptype;
}


void RIOQueue::enque(Packet* pkt)
{

    /*
     * if we were idle, we pretend that m OUT packets arrived during
     * the idle period.  m is set to be the ptc times the amount
     * of time we've been idle for
     */

    int m = 0;
    if (idle_) {
        double now = Scheduler::instance().clock();
        /* To account for the period when the queue was empty. */
        idle_ = 0;
        m = int(OUTedp_.ptc * (now - idletime_));
    }

    /* get the DSCP field */
    hdr_ip* iph=hdr_ip::access(pkt);
    /*
     * Run the estimator with either 1 new packet arrival, or with
     * the scaled version above [scaled by m due to idle time]
     * (bcount_ maintains the byte count in the underlying queue).
```

```
         * If the underlying queue is able to delete packets without
         * us knowing, then bcount_ will not be maintained properly!
         */
      #ifdef hannan_debug
      //printf("Tag = %d\n", iph->prio_);
      #endif

        tag_ = iph->prio_;

      /* since OUT packets are based on the total queue length, so we update OUT average every time a
packet comes in */
          run_estimator(qib_ ? bcount_ : q_->length(), m + 1, &OUTedp_, &OUTedv_);

              if (iph->prio_ == IN)
        /* extimate the average queue size based on the # of IN packetrs present in the queue */
              run_estimator(qib_ ? bcount_ : INnum_, 1, &INedp_, &INedv_);
       /*   else if (iph->prio_ == OUT)
      */
        /* estimate the average queue size based on the total # of OUT packets
        present in the queue */
       /*
           run_estimator(qib_ ? bcount_ : q_->length(), m + 1, &OUTedp_, &OUTedv_);
      */

        /*
         * count and count_bytes keeps a tally of arriving traffic
         * that has not been dropped (i.e. how long, in terms of traffic,
         * it has been since the last early drop)
         */
        hdr_cmn* ch = (hdr_cmn*)pkt->access(off_cmn_);

        if (iph->prio_ == IN) {
          ++INedv_.count;
          INedv_.count_bytes += ch->size();
        }
        else if (iph->prio_ == OUT) {
          ++OUTedv_.count;
          OUTedv_.count_bytes += ch->size();
        }

        /* decide drop type of the packet */
        int droptype = DTYPE_NONE;

        if (iph->prio_ == IN)
          droptype = decide_droptype(pkt, &INedp_, &INedv_);
        else if (iph->prio_ == OUT)
          droptype = decide_droptype(pkt, &OUTedp_, &OUTedv_);

         /* forced drop, or not a drop: first enqueue pkt */
          q_->enque(pkt);
          bcount_ += ch->size();

              // if is IN packet, increase the INnum_

             if (iph->prio_ == IN) INnum_++;
```

```
                /* drop a packet if we were told to */

        //Main part in the implementation of SED, SED/RIO and SED with timestamps

        int count1 = 1;
        int count2;
        int param = 0;

        double local_time;

        /get the local time

        local_time = Scheduler::instance().clock();

        //this loop examines if a packet is delayed more than 400 ms (8000*0.400=3200)

        for (count2 = 1; count2 <= (q_->length()-1); count2++) {
                if  ( ( (local_time*8000) - hdr_cmn::access(q_->lookup(count2))->timestamp() ) >
3200  )  {
                        count1 = count2;
                }
        }

        double InTh1_ = 120;
        double InTh2_ = 140;
        double InTh3_ = 160;

        double OutTh1_ = 50;
        double OutTh2_ = 70;
        double OutTh3_ = 80;

        if (count1 == 1) {   //examine if there is packet to drop, if no late pkt exists

                if (iph->prio_ == IN) {
                        if ( (q_->length()-1) >= InTh1 && (q_->length()-1) < InTh2 ) {

                                for (count2 = (q_->length()-1); count2 >= 1; count2--) {
                                        if  ( (hdr_rtp::access(q_->lookup(count2))->seqno() % 20)
== 0
                                                && iph->prio_ == (hdr_ip::access(q_-
>lookup(count2))->prio() )          )  {
                                                count1 = count2;
                                        }
                                }
                        }else if ( (q_->length()-1) >= InTh2 && (q_->length()-1) <= InTh3 )   {

                                for (count2 = (q_->length()-1); count2 >= 1; count2--) {
                                        if  ( (hdr_rtp::access(q_->lookup(vic))->seqno() % 10) == 0
                                                && iph->prio_ == (hdr_ip::access(q_-
>lookup(count2))->prio() )           )  {
                                                count1 = count2;
                                        }
                                }
                        } else if ((q_->length()-1) > InTh3 ) {
                        for (count2 = (q_->length()-1); count2 >= 1; count2--) {
```

```
                                        if  ( (hdr_rtp::access(q_->lookup(count2))->seqno() % 5)
== 0
                                                && iph->prio_ == (hdr_ip::access(q_-
>lookup(count2))->prio() )   )  {
                                                count1 = count2;
                                        }
                                }
                        }
                } else if (iph->prio_ == OUT) {
                        if ((q_->length()-1) >= OutTh1 && (q_->length()-1) < OutTh2 ) {
                                for (count2 = (q_->length()-1); count2 >= 1; count2--) {
                                        if  ( (hdr_rtp::access(q_->lookup(count2))->seqno() % 16)
== 0
                                                && iph->prio_ == (hdr_ip::access(q_-
>lookup(count2))->prio() )   )  {
                                                count1 = count2;
                                        }
                                }
                        }else if ((q_->length()-1) >= OutTh2 && (q_->length()-1) <= OutTh3 )   {

                                for (count2 = (q_->length()-1); count2 >= 1; count2--) {
                                        if  ( (hdr_rtp::access(q_->lookup(count2))->seqno() % 8)
== 0
                                                && iph->prio_ == (hdr_ip::access(q_-
>lookup(count2))->prio() )   )  {
                                                count1 = count2;
                                        }
                                }
                        } else if ((q_->length()-1) > OutTh3) {
                        for (count2 = (q_->length()-1); count2 >= 1; count2--) {
                                        if  ( (hdr_rtp::access(q_->lookup(count2))->seqno() % 4)
== 0
                                                && iph->prio_ == (hdr_ip::access(q_-
>lookup(count2))->prio() )   )  {
                                                count1 = count2;
                                        }
                                }
                        }
                }

        } //end if for loop count1=1


        if (count1 != 1) {   //if there is a droppable packet...
                pkt = q_->lookup(count1);
                q_->remove(pkt);
                // if is IN packet, decrease the INnum_

                hdr_ip* iph=hdr_ip::access(pkt);
                if (iph->prio_ == IN) INnum_--;
                        bcount_ -= ((hdr_cmn*)pkt->access(off_cmn_))->size();
                        drop(pkt);
        } else {
                if ( (q_->length()-1) > InTh3 && iph->prio_ == IN ) {        //droptype ==
DTYPE_FORCED) {
                        //pkt = q_->lookup(q_->length() -1);
```

```
                    //pkt = q_->lookup(count1);
                    q_->remove(pkt);
                    param == 1;
            }
        else if ( (q_->length()-1) > OutTh3 && iph->prio_ == OUT ) {
                  // if is IN packet, decrease the INnum_
                    q_->remove(pkt);
                    param == 1;
            }
            if (param == 1 ) {  //if it is forced drop drop pkt & fix counter
                    hdr_ip* iph=hdr_ip::access(pkt);
                    if (iph->prio_ == IN) INnum_--;
                    bcount_ -= ((hdr_cmn*)pkt->access(off_cmn_))->size();

                    drop(pkt);
            }
}

   return;
}

int RIOQueue::command(int argc, const char*const* argv)
{
   Tcl& tcl = Tcl::instance();
   if (argc == 2) {
      if (strcmp(argv[1], "reset") == 0) {
         reset();
         return (TCL_OK);
      }
      if (strcmp(argv[1], "early-drop-target") == 0) {
         if (de_drop_ != NULL)
            tcl.resultf("%s", de_drop_->name());
         return (TCL_OK);
      }
   } else if (argc == 3) {
      // attach a file for variable tracing
      if (strcmp(argv[1], "attach") == 0) {
         int mode;
         const char* id = argv[2];
         tchan_ = Tcl_GetChannel(tcl.interp(), (char*)id, &mode);
         if (tchan_ == 0) {
            tcl.resultf("RIO: trace: can't attach %s for writing", id);
            return (TCL_ERROR);
         }
         return (TCL_OK);
      }
      // tell RIO about link stats
      if (strcmp(argv[1], "link") == 0) {
         LinkDelay* del = (LinkDelay*)TclObject::lookup(argv[2]);
         if (del == 0) {
            tcl.resultf("RIO: no LinkDelay object %s",
               argv[2]);
            return(TCL_ERROR);
         }
         // set ptc now
         link_ = del;
```

```
            INedp_.ptc = link_->bandwidth() /
               (8. * INedp_.mean_pktsize);
            OUTedp_.ptc = link_->bandwidth() /
               (8. * OUTedp_.mean_pktsize);
            return (TCL_OK);
         }
         if (strcmp(argv[1], "early-drop-target") == 0) {
            NsObject* p = (NsObject*)TclObject::lookup(argv[2]);
            if (p == 0) {
               tcl.resultf("no object %s", argv[2]);
               return (TCL_ERROR);
            }
            de_drop_ = p;
            return (TCL_OK);
         }
         if (!strcmp(argv[1], "packetqueue-attach")) {
            delete q_;
            if (!(q_ = (PacketQueue*) TclObject::lookup(argv[2])))
               return (TCL_ERROR);
            else {
               pq_ = q_;
               return (TCL_OK);
            }
         }
      }
   return (Queue::command(argc, argv));
}

/*
 * Routine called by TracedVar facility when variables change values.
 * Currently used to trace values of avg queue size, drop probability,
 * and the instantaneous queue size seen by arriving packets.
 * Note that the tracing of each var must be enabled in tcl to work.
 */

void
RIOQueue::trace(TracedVar* v)
{
   char wrk[500], *p;

   if (((p = strstr(v->name(), "in_ave")) == NULL) &&
           ((p = strstr(v->name(), "out_ave")) == NULL) &&
      ((p = strstr(v->name(), "in_prob")) == NULL) &&
      ((p = strstr(v->name(), "out_prob")) == NULL) &&
      ((p = strstr(v->name(), "curq")) == NULL) &&
      ((p = strstr(v->name(), "tag")) == NULL)) {
      fprintf(stderr, "RIO:unknown trace var %s\n",
         v->name());
      return;
   }

   if (tchan_) {
      int n;
      double t = Scheduler::instance().clock();
      // XXX: be compatible with nsv1 RIO trace entries
// added for debug hannan
```

```
            if (t ==125) {
                    printf("Here time =125\n");
            }
        if ((p = strstr(v->name(), "curq_")) != NULL) {
          sprintf(wrk, "CURQ %g %d",  t, int(*((TracedInt*) v)));
            } else if ((p = strstr(v->name(), "in_ave_")) != NULL) {
          sprintf(wrk, "INAVE %g %g", t, double(*((TracedDouble*) v)));
        } else if ((p = strstr(v->name(), "out_ave_")) != NULL) {
          sprintf(wrk, "OUTAVE %g %g", t, double(*((TracedDouble*) v)));
        } else if ((p = strstr(v->name(), "in_prob1_")) != NULL) {
          sprintf(wrk, "INPROB %g %g", t, double(*((TracedDouble*) v)));
        } else if ((p = strstr(v->name(), "out_prob1_")) != NULL) {
          sprintf(wrk, "OUTPROB %g %g", t, double(*((TracedDouble*) v)));
        } else if ((p = strstr(v->name(), "tag")) != NULL) {
          sprintf(wrk, "DHCP %g %d", t, int(*((TracedInt*) v)));
        }
        n = strlen(wrk);
        wrk[n] = '\n';
        wrk[n+1] = 0;
        (void)Tcl_Write(tchan_, wrk, n+1);
    }
    return;
}

/* for debugging help */
void RIOQueue::print_edp(edp* edp_)
{
    printf("mean_pktsz: %d\n", edp_->mean_pktsize);
    printf("bytes: %d, wait: %d, setbit: %d\n",
          edp_->bytes, edp_->wait, edp_->setbit);
    printf("minth: %f, maxth: %f\n", edp_->th_min, edp_->th_max);
    printf("max_p_inv: %f, qw: %f, ptc: %f\n",
          edp_->max_p_inv, edp_->q_w, edp_->ptc);
    printf("qlim: %d, idletime: %f\n", qlim_, idletime_);
    printf("=========\n");
}

void RIOQueue::print_edv(edv* edv_)
{
    printf("v_a: %f, v_b: %f\n", edv_->v_a, edv_->v_b);
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. *"Operational Requirements Document (ORD) for Joint Tactical Radio System (JTRS),"* JTRS Joint Program Office, 23 March 1998.

2. J.Jubin and J.D.Tornow, "The DARPA Packet Radio Project," in *Proceedings* of *IEEE*, 1987.

3. Corson, Scott S., Macker, J., "Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," RFC 2501, January 1999.

4. Rappaport, Theodore S., *Wireless Communications – Principles and Practice*, Upper Saddle River, NJ: Prentice Hall, 1996.

5. Xu S.,Saadaei T., "Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks?," in *IEEE Communications Magazine*, Vol 39 Issue 6, pg 130-137, June 2001.

6. Li J., Blake C., De Couto D., Lee H., Morris R., "Capacity of Ad Hoc Wireless *Networks," in* the *7th ACM International Conference on Mobile Computing and Networking, Rome, Italy, July 2001.*

7. H. Holland and N.H Vaidya, "Analysis of TCP performance over mobile ad-hoc networks," *in Proceedings of IEEE/ACM Mobicom99*, pages 219-230, Seattle, WA, USA, August 1999.

8. IEEE, *"IEEE std 802.11 – wireless LAN medium access control (MAC) and physical layer (PHY) specifications,"* 16 November, 1997.

9. Hubaux J., Thomas G., Boudec J., Veterli M., "Towards Self-Organized Mobile Ad Hoc Networks: The Terminodes Project", in *IEEE Communications Magazine*, Vol 39 Issue 1, pg 118-124, January 2001.

10. Vaidya, Nitin H., *"Mobile Ad Hoc Networks; Routing, MAC, and Transport Issues,"* MobiComm Tutorial, 15 July, 2000, pg 1- 431.

11. Haas, Zygmunt J, Pearlman, Marc R., "The Zone Routing Protocol (ZRP) for Ad Hoc Networks," Internet Draft, draft-ietf-manet-zone-zrp-02.txt, June 1999.

12. Gerla, M., Lee, S. J., Toh, C. K., "A Simulation Study of Table-Driven and On-Demand Routing Protocols for Mobile Ad Hoc Networks," *IEEE Network*, Vol 13 Issue 4, pg 48-54, Jul-Aug 1999.

13. Royer E., Toh C., "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," in *IEEE Personal Communications*, Vol 6 Issue 2, pg 46-55, April 1999.

14. Mattos S. L., "Quality of Service Analysis in Mobile Ad-hoc Networks," Engineer's Thesis, Naval Postgraduate School, Monterey, California, March 2001.

15. Theriot, Ty, "Simulation and Performance Analysis of the AODV protocol for Tactical Mobile Ad-hoc Networks," Master's Thesis, Naval Postgraduate School, Monterey, California, December 2000.

16. Shea, Kevin, "Simulation and Performance Analysis of the ZRP protocol for Tactical Mobile Ad-hoc Networks," Master's Thesis, Naval Postgraduate School, Monterey, California, September 2000.

17. Stallings William., *High Speed Networks TCP/IP and ATM Design Principles*, Upper Saddle River, NJ: Prentice Hall, 1998

18. Information Sciences Institute, University of Southern California, "Transport Control Protocol, DARPA Internet Program, Protocol Specification," IETF RFC 793, September 1981.

19. Jacobson V., LBL, Braden R., ISI, Borman D., Cray Researsh, "TCP extensions for High Performance," ITEF RFC 1323, May 1992.

20. Mahdavi J., PSC, Floyd S., LBNL, Romanow A., Sun Microsystems, "TCP Selective Acknowledgment Options," IETF RFC 2018, October 1996.

21. Postel J., ISI, "User Datagram Protocol," IETF RFC 768, August 1980.

22. Audio-Video Transport Working Group, Schulzrinne H., GMD Fokus, Casner, Precept Software, Inc., Frederick R., Xerox Palo Alto Research Center, Jacobson V., Lawrence Berkeley National Laboratory, "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 1889, January 1996.

23. Casner S., Cisco Systems, Jacobson V., "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links," IETF RFC 2508 February 1999.

24. Cisco white paper, "Quality of Service (QoS) Networking," http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm

25. White paper, "QoS protocols and Architectures," July 1999, www.qosforum.com

26. R.Braden, D.Clark, and S.Shenker, "Integrated Services in the Internet Architecture - an Overview," IETF RFC1633, June 1994.

27. Blake, S., "An architecture for Differentiated Services", IETF RFC2475, December 1998.

28. Braden B., Clark D., Crowcroft J., Davie B., Deering S., Estrin D., Floyd S., Jacobson V., Minshall G., Partridge C., Peterson L., Ramakrishnan K., Shenker S., Wroclawski J., Zhang L., "Recommendations on Queue Management and Congestion Avoidance in the Internet," IETF RFC 2309, April 1998.

29. Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance," Lawrence Berkeley Laboratory, University of California, *IEEE/ACM Transactions on Networking*, August 1993.

30. Information Sciences Institute, University of Southern California, "Internet Protocol, DARPA Internet Program, Protocol Specification," IETF RFC 791, September 1981.

31. S Deering, Cisco, R. Hidden, Nokia, "Internet Protocol, version 6 (Ipv6). Specification, "IETF RFC 2460, December 1998.

32. C. Flower, "Definitions of Management Objects for the DS0 and DS0 Bundle Interface Type," IETF RFC 2494, January 1999.

33. Nortel, "Packet Loss and Packet Loss concealment," Technical Brief, February, 2000.

34. Cisco, "Managing Voice Quality with Cisco Voice Manager (CVM) and Telenate," White paper, March 2000.

35. C.Perkins, O.Hodson, V.Hardman, "A survey of packet-loss recovery techniques for streaming audio," in *IEEE Network*, August 1998.

36. K. Sriram, Y.T. Wang, "Voice over ATM using bit dropping: Performance and call admission control," in *IEEE J. Select. Areas Commun*., vol. 17, no. 1, pp. 18-28, Jan. 1999.

37. Campbell, J. P., Jr., T. E. Tremain, and V. C. Welch. "The Federal Standard 1016 4800 bps CELP Voice Coder" *Digital Signal Processing 1*, no. 3, pg. 145-155, 1991.

38. Spanias A., Painter R., "FS 1016 Vocoder in Matlab," http://www.cysip.com/FS1016_FORM.htm

39. International Telecommunication Union (ITU), "Transmission Impairments Due to Speech Processing," Recommendation G.113, February 2001.

40. International Telecommunication Union (ITU), "Terms and definitions related to quality of service and network performance including dependability," Recommendation E.800, August 1994.

41. International Telecommunication Union (ITU), "One-way transmission time," Recommendation G.114, May 2000.

42. K. Fall and K. Varadhan, "ns Notes and Documentation," VINT Project, UC-Berkeley and LBNL, August 2000.

43. H. Xiao, W.K.G. Seah, A.Lo., and K.C.Chua., "A Flexible Quality of Service Model for Mobile Ad-Hoc Networks," in *Proceedings of IEEE VTC2000-spring*, Tokyo, Japan, May 2000.

44. D.D. Clark and W. Fang, "Explicit Allocation of Best-effort Packet Delivery Service," in *IEEE/ACM Trans. On Networking*, 6(4):362-373, August 1998.

45. Cansever, Derya H., Levesque, Allen H., Michelson, Arnold M. "Quality of Service Support in Mobile Ad-Hoc Networks," in *Proceedings of Military Communications Conference, MILCOM99*.

46. Lee, S.B., Campbell, A.T., "INSIGNIA: In-band signaling support for QoS in mobile ad-hoc networks," *5th Int. Workshop on Mobile Multimedia Communications (MoMuc)*, Berlin, Germany, October 1998

47. H. Xiao, W.K.G. Seah, A.Lo., and K.C.Chua., "A Flexible Quality of Service Model for Mobile Ad-Hoc Networks," in *Proceedings of IEEE VTC2000-spring*, Tokyo, Japan, May 2000.

48. Le Faucet, "MPLS Support of Differentiated Services," draft-ietf-mpls-traffic-eng-01.txt, June 1999.

49. Horlait E., Rouhana N., "Differentiated Services and Integrated Services use of MPLS," University Pierre and Marie Curie – France, June 2001.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Engineering and Technology Curricular Office, Code 34
   Naval Postgraduate School
   Monterey, California

4. Chairman, Code EE
   Naval Postgraduate School
   Monterey, California

5. Dr. Murali Tummala, Code EC/Tu
   Department of Electrical and Computer Engineering
   Naval Postgraduate School

6. Dr. Robert Ives (LCDR-USN), Code EC/Ir
   Department of Electrical and Computer Engineering
   Naval Postgraduate School

7. Dr. Richard North
   SPAWAR Systems Center
   San Diego, California

8. Dr. Robert E. Parker (LCDR-USN)
   SPAWAR Systems Center
   San Diego, California

7. Hellenic Navy
   General Staff
   B2 Directorate
   Athens, Greece

8. Leonidas Fountanas
   Lieutenant, Hellenic Navy
   Athens, Greece